ScienceDirect

Computer Science Review

Survey

A survey on parallel and distributed multi-agent systems for high performance computing simulations

*Alban Rousset**, *Bénédicte Herrmann*, *Christophe Lang*, *Laurent Philippe*

Femto-ST Institute CNRS/UFC – Faculté des Sciences et des Techniques, 16 Route de Gray 25030 Besançon cedex, France

HIGHLIGHTS

- We survey Parallel and Distributed Multi-Agents Systems for HPC.
- We analyze the platform properties for distribution support.
- We have implemented a reference model to assess the impact of the properties on the simulation development.
- We assess the performance impact on the simulation scalability.

ARTICLE INFO

Article history:
Received 8 October 2015
Received in revised form
30 June 2016
Accepted 2 August 2016

Keywords:
Multi-agent simulation
Parallelism
MAS
High performance computing

ABSTRACT

Simulation has become an indispensable tool for researchers to explore systems without having recourse to real experiments. Depending on the characteristics of the modeled system, methods used to represent the system may vary. Multi-agent systems are often used to model and simulate complex systems. In any cases, increasing the size and the precision of the model increases the amount of computation, requiring the use of parallel systems when it becomes too large. In this paper, we focus on parallel platforms that support multi-agent simulations and their execution on high performance resources as parallel clusters. Our contribution is a survey on existing platforms and their evaluation in the context of high performance computing. We present a qualitative analysis of several multi-agent platforms, their tests in high performance computing execution environments, and the performance results for the only two platforms that fulfill the high performance computing constraints.

Contents

* *Corresponding author.*
E-mail addresses: alban.rousset@femto-st.fr (A. Rousset), benedicte.herrmann@femto-st.fr (B. Herrmann), christophe.lang@femto-st.fr (C. Lang), laurent.philippe@femto-st.fr (L. Philippe).

http://dx.doi.org/10.1016/j.cosrev.2016.08.001
1574-0137/

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001

1. Introduction

In the field of simulation, to improve the confidence in a result, we often need to overcome the limits of a model, for instance by increasing its size (simulate larger systems) or its accuracy (smaller discretization of the system). Increasing the size of a model or improving its accuracy has however a direct impact on the amount of computations necessary to animate the model. More computing resources are thus needed and centralized systems are often no longer sufficient to run these simulations. The use of parallel resources allows us to overcome the resource limits of centralized systems and thus to increase the size and the accuracy of the simulated models.

There are several ways to model a system [1]. For example, the time behavior of a large number of physical systems is based on differential equations. In this case the discretization of a model allows, most of the time, its representation as a linear system. It is then possible to use existing parallel libraries to take advantage of many computing nodes and run large simulations. On the other hand it is not possible to model every time-dependent system with differential equations. This is for instance the case of complex systems as defined in [2] where the complexity of the dependencies between the phenomena that drive the entity behaviors makes it difficult to define a global law modeling the entire system. For this reason multi-agent systems (MAS) are often used to model complex systems. MAS are based on an algorithmic description of individuals, agents, that simulate the expected behavior. Then the platform is in charge of animating the model, i.e. running the agent behaviors, either using a periodic time discretization, time steps, or a list of scheduled events. MAS thus propose a bottom-up modeling approach as opposed to the top-down approach of formal models. From the viewpoint of increasing the size or accuracy of simulations, multi-agent systems are constrained to the same rules as other modeling techniques.

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001

There however exists less support for parallel execution of multi-agent models compared to the large set of parallel libraries available for linear systems. The interest for parallel multi-agent platforms has recently increased. This is for instance the case in the simulation of individual displacements in a city/urban mobility [3,4]. By offering more resources, parallel platforms allow to run larger agent simulations and thus to obtain results or behaviors that were not possible to obtain with smaller number of agents.

In this article, we focus on multi-agent simulation platforms that run on parallel distributed environments such as high performance clusters and supercomputers in computing centers. The contribution of the article is a survey on parallel distributed multi-agent simulation platforms and a performance assessment of the HPC (High Performance Computing) compliant platforms. This survey is based on an extensive bibliographical work done to identify the existing platforms. We assess these platforms from both qualitative and performance view points and propose a classification of the platforms from their description in articles. Last, we test the compliance of the platforms to HPC execution environments and we compare performance of the only two platforms (i.e. FLAME and RepastHPC) that fulfill the requirements. To reach this goal we run them on a HPC cluster with a representative model.

The article is organized as follows. In Section 2 we introduce the context of agents and multi-agent systems (MAS) in general. In Section 3, we give the context of parallel distributed multi-agent systems (PDMAS), an overview of HPC platforms and the constraints set when running applications. Then, in Section 4, we present the different multi-agent platforms found in our bibliographical research. In Section 5, we describe the method used to classify platforms and we propose an extensive study of their qualitative properties. In Section 6, we present the qualitative comparison of the different PDMAS followed, in Section 7, by the benchmark based on the implemented model. We finish the paper with conclusion and future work.

2. MAS based simulations

Similarly to the object concept in object oriented development, the agent concept has been proposed to ease the development of dedicated software. The concept of agent has been extensively studied for several years and in different domains. One of the first definition of the agent concept is due to Ferber [5]: "*An agent is a real or virtual autonomous entity, operating in an environment, able to perceive and act on it, which can communicate with other agents, which exhibits an independent behavior, which can be seen as the consequence of his knowledge, its interactions with other agents and goals it need to achieved*" and has been generalized by Woolridge [6]. It is used in a large number of software related fields as robotics and artificial intelligence, but also in more general fields such as psychology [7] or biology [8].

The agent concept covers different practical implementations. The more widespread ones are management agents and simulation agents. Management agents are implemented as autonomous software entities that, for instance, manage a hardware platform. Management agents are, for example, used in the SNMP network management protocol to gather real time information on the platform state. In the following we refer to implementations based on management agent as agent based applications. On the other hand a simulation agent intends to simulate a real behavior in a simulation software. It is usually a piece of software code that encapsulates the behavior of an autonomous entity. Simulation agents are used in multi-agent models to simulate a large set of entities. Simulating a system using the multi-agent paradigm aims at identifying global behaviors or tendencies from individual behaviors. This is a bottom-up approach of representing a system composed of individuals. In the following we will refer to for the implementation of a model with agents as multi-agent simulations. In this paper we concentrate on multi-agent simulations rather than on agent based applications.

Developing a multi-agent simulation hence requires the definition of algorithms that simulate the individual behaviors. These algorithms are then run by a simulator to reproduce the individual behaviors on the simulation data. Two main schedule approaches are used by the platforms to animate the model. The "time step" schedule approach assumes a regular discretization of the simulated time in steps. At each time step the activatable behaviors of the agents are run. In the "event based" schedule approach, agents are activated by events scheduled on the time scale. Note that time driven simulations can be developed with event driven platforms by programming periodical events that run the agent behavior. Representative data, outputs of the simulation, are computed by animating the model, i.e. iterating on time, as for a large range of simulation software.

As stated, in the definition, agents operate on an environment and interact with each other. In the multi-agent simulation context, the environment usually represents data shared by agents. For instance an environment may represent a geographic area in which the agents move. From an implementation point of view this implies that the simulator provides an access to these shared data. This could be either implemented by defining global data or with specific access procedures. Interactions between agents may represent direct interactions as wolves eating sheep or social interactions when two persons know each other. These interactions can either be done through the environment or through direct calls.

A multi-agent system (MAS) platform, is a generic platform that provides the necessary support to run simulations based on agents. MAS platforms are designed to make the development and assessment of multi-agent models easier. Among some well known platforms we can cite Repast Simphony [9], Mason [10], NetLogo [11] and Gama [12]. Their interfaces provide services such as agent life cycle management, communication between agents, agent perception or environment management. The properties of MAS largely differ depending on the models they target: programming language, ease of development, user interface, etc.

There already exist several works proposing a survey on multi-agent platforms [13–16]. These works however only describe the basic properties of the studied MAS and do not define a true comparison framework. Only [17] proposes a classification based on criteria.

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001

3. PDMAS and HPC

Most of MAS platforms do not natively implement a support to run multi-agent simulations in parallel. Possible approaches to distribute or parallelize a simulation include the development of a dedicated model as in [18] or the implementation of a wrapper from scratch [19]. These approaches are however complex as they require parallel programming skills while most of multi-agent models are developed by non-specialist programmers. Although some works [20,21] propose approaches to facilitate the distribution of multi-agent systems, tuning a parallel simulation, agent or not, to efficiently run it on a parallel machine requires specific skills. For this reason it is easier to rely on a Parallel and Distributed MAS (PDMAS). A PDMAS provides the necessary support to easily use and operate agent simulations on parallel resources. Note that, as for other parallel simulations, a parallel multi-agent model, with specific optimizations and correctly tuned for a given parallel machine, will probably reach the best performances.

Compared to the large number of MAS, only few Parallel and Distributed MAS provide a native support for parallel execution of models. This support usually includes the collaboration between executions on several physical nodes, the distribution of agents between nodes, the communication between agents, the synchronization between several processing nodes, and so on. One of the objectives of this survey is to highlight the properties of the existing PDMAS regarding parallelism.

During our analysis of the literature, we did not find any survey about parallel multi-agent platforms except the paper written by Coakley et al. [22]. This work surveys PDMAS based on qualitative criteria such as the implementation language but it does not provide any performance comparison of the studied platforms. For this reason we propose in this paper a full comparison of the existing platforms and of their compliance with HPC platforms. The method used is based on four steps. First we gathered information on existing platforms with search engines, see Section 4. Second, from the papers describing these platforms, we did a quantitative analysis based on several criteria, see Section 5. Third we implemented a reference model to assess the model development and validate the quantitative criterion, see Section 6. Fourth, we tested the performance of the model on a HPC platform, see Section 7. Note that the quantitative and development analyses provide a larger scope than the performance section as they consider platforms and criteria not only linked to HPC.

Access to distributed or parallel resources is nowadays becoming more and more common for scientists who want to run large models. Several types of hardware platforms can be used to improve the performance of an application and today the most widespread parallel platforms are shared memory machines, distributed memory machines and many-core cards (Graphical Processing Unit or GPU and Xeon Phi). Each kind of machine has its own programming model. On the one hand shared memory platforms propose a general purpose programming model but they are limited in number of cores. On the other hand many-core cards propose a high number of cores but must be programmed with dedicated languages. Distributed memory machines and the MIMD (Multiple Instructions Multiple Data) programming model are the most used for large simulations because they propose in the same time a general programming model and they can scale up to several thousands of cores. In the PDMAS context, they can support large models and simulations that scale up in terms of number of agents.

Although there does not exist a standardized definition, HPC resources usually refer to a set of computers tightly coupled by a high speed network, a cluster or a supercomputer. They are specialized platforms that set limits to the application to be run. Due to their price HPC resources are usually available in computing centers where they are shared among users through a resource manager. In these computing centers applications are run in batch mode which prevents the use of interfaces and of other interactive behaviors. On the other hand the resources provide specialized hardware as high speed networks (e.g. InifiniBand) and powerful cores that can support the execution of large models. In the remainder of the paper, and in particular in the performance evaluation section, we focus on HPC resources.

4. Existing PDMAS platforms

As already stated we started this study by a bibliographical search, using keywords on search engines and following links cited in the studied articles. This bibliographical search allowed us to establish a first list of ten existing platforms or projects of PDMAS platforms. To our knowledge this list is complete and their is no other available and functional platform that provides a generic support for PDMAS. We succinctly present each of these platforms in the following.

D-MASON (Distributed Mason) [23,24] is developed by the ISISLab at University of Salerno. D-MASON is the distributed version of the Mason multi-agent platform. The authors choose to develop a distributed version of Mason to provide a solution that overcome the limitations on maximum number of agents without rewriting already developed simulations. D-MASON uses ActiveMQ JMS (Java Messaging Service) and MPI (Message Passing Interface) as a base to implement communications and distribution. D-MASON uses the Java language to implement the agent model.

Flame [22] is developed by the University of Sheffield. Flame was designed to allow a wide range of agent models. Flame provides a formal framework that can be used to create models and tools. Implementing a Flame simulation is based on the definition of X-Machines [25] which are defined as finite state automaton with memory. From this specification the framework can automatically generate simulation programs with parallelization based on MPI. In addition, agents can send and receive messages at the input and the output of each state.

Jade [26] is developed by the Telecom laboratory of Italia. The aims of Jade are to simplify the implementation of distributed multi-agent models across a FIPA compliant [26] middle-ware. The platform provides a set of tools that makes the debugging and the deployment phases easier. The platform can be distributed across multiple computers and its configuration can be controlled from a remote GUI. Agents are implemented in Java while the communications may use

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001

several transport layers as Java-RMI, HTTP or IIOP to support FIPA compliant messages.

Pandora (Distributed Mason) [27] is developed by the Supercomputing center of Barcelona. It is explicitly programmed to allow the execution of scalable multi-agent simulations. According to the literature, Pandora is able to treat thousands of agents with complex actions. Pandora also provides a support for a geographic information system (GIS) in order to run simulations where agents have spatial coordinates. Pandora uses the C++ language to define and to implement the agent models. For the communications, Pandora automatically generates MPI code from the Pandora library.

RepastHPC [28] is developed by the Argonne National Laboratory. It is part of a series of multi-agent simulation platforms: RepastJ and Repast Simphony. RepastHPC is specially designed for high performance environments. RepastHPC uses the same concepts (grid, network) as the core of Repast Simphony to represent agent environment but they are adapted to parallel environments. The C++ language is used to implement agent models but the ReLogo language, a derivative of the NetLogo language, can also be used. The RepastHPC platform relies on MPI using the Boost library [29] For the communications.

PDES-Mas [30,31] is developed by Distributed Systems Lab at the University of Birmingham. PDES-Mas is a framework and a system (simulation kernel) for the distributed simulation of agent-based systems. PDES-MAS adopts a standard discrete event simulation approach with optimistic synchronization. Each agent is modeled as an Agent Logical Process (ALP). The environment, called the shared state, is maintained by a dynamically and transparently reconfigured tree-structured set of additional logical processes, the Communication Logical Processes (CLP), which cluster agent models and shared state. PDES-Mas uses MPI as a communication layer.

SWAGES [32] is developed by the University of Notre Dame (USA). SWAGES provides automatic dynamic parallelization and distribution of simulations in heterogeneous computing environments to minimize simulation times. This framework is generally divided into server-side and client-side components. SWAGES allows to develop agent simulations with any of the available programming languages in Poplog, adding external function calls to C or Java if necessary. The communication library used is XML-RPC and SSML.

Ecolab [33] is developed by the University of New South Wales. Ecolab is an agent based modeling systems for C++ programmers. Ecolab is strongly influenced by the design of Swarm. Ecolab uses MPI as a communication layer.

MACE3J [34] is developed by the university of Illinois. MACE 3J is a Java-based MAS simulation, integration, and development test-bed. MACE3J runs on multiprocessor workstations and in large multiprocessor cluster environments. The MACE3J design is multi-grain, but gives special attention to simulating very large communities of large-grain agents. It exhibits a significant degree of scalability. No information has been found on the used communication layer.

ABM++ is developed by the Los Alamos National Laboratory. The ABM++ framework allows to implement agent based models using C++ and provide an interface to access the MPI communication layer. It provides the necessary functions to run models on distributed memory architectures, as the possibility to serialized objects into message buffers to move them between nodes.

5. Qualitative analysis

Using the documentation supplied for each platform we made a qualitative analysis of their properties and of the support provided to implement models. This qualitative analysis first concentrates on general properties that are useful when developing a model, as the programming language or the way agents are synchronized. We also analyze the support provided by the platforms to implement parallel models. We however do not assess the support provided in this section and we limit our analyze to the information given in available documentations. From these information we select the platforms that fall in the scope of our study, i.e. multi-agent simulations.

5.1. *Method*

We present here the criteria used to compare and analyze each platform. Three sets of criteria are defined to assess the platforms.

The first set mainly concerns agent development and properties of agent entities in the platform. These criteria intend to give information on how agents are implemented and represented. With these criteria we aim at characterizing the programming model proposed by the platform. The used criteria are:

1. Programming language: the language used to develop models. Note that some platforms propose a different language for the simulation definition (interactions, agent state) than for the agent behavior definition.
2. Agent representation: the concept used to represent agents in the programming language. Every platform, as a support to multi-agent models, proposes the agent paradigm. As most platforms are implemented with an object oriented language this paradigm is usually based on the object paradigm. Depending on the platform implementation, agents may be represented not only by an object but also by another set of data that complete the agent description.
3. Agent state: the agent state is usually made of the agent local data. This criterion gives the agent state representation in platform.
4. Agent behavior: the behavior is the dynamic part of the agent. The representation of the agent behavior may take different shapes which are of importance when implementing a model.
5. Agent identification: agents are identified in a simulation. This identification is necessary to send messages or to interact with other agents. As agents will be distributed in a parallel environment, a suitable agent identification is important.

The second set of criteria groups the general properties of the platforms regarding the simulation and the guaranties provided. These criteria are:

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001

1. Agent synchronization: multi-agent simulations are driven by a scheduler that animates the model, i.e. the scheduler is in charge of running the agent behavior on the events. As previously presented two main time discretizations are usually considered: time driven and event driven.
2. Simultaneous events: in sequential MAS events and agent behaviors are processed one after another. In parallel platforms, like PDMAS, several runs are done simultaneously.
3. Reproducibility: simulating a model must be a deterministic process and we should always get the same result with a given set of parameters. Reproducibility is a mandatory property in MAS to be confident in the results. Note however that this criterion can reach its limit in a parallel context. Running simulations on different machines with different speeds make it difficult to implement an absolute reproducibility: (1) the order of agent activation may differ when they do not run on the same node as nodes are not synchronized, (2) the order in which messages are delivered to agents depends on the network propagation of these messages and on the relative advance of some nodes on the agent schedule.
4. Random numbers: agent behaviors often rely on random choices and a random number generator is usually provided with multi-agent platforms. The properties of the random number generator are of importance when choosing a multi-agent platform. Note also that the random number generation may introduce some bias in the reproducibility of simulations [35].

The third set of criteria groups the platform characteristics regarding parallelism. As this survey considers parallelism support in platforms these criteria are of importance. There are several ways of implementing a PDMAS and the implementation choices impact the platform scope and quality. While centralized MAS are based on one instance of the platform, PDMAS are composed of several platform instances, one per node usually, that collaborate to represent the global system. This collaboration lays on synchronizations and communications between the instances. Properties of the considered PDMAS and Facilities provided to the model developer depend on the way these collaborations are realized. The considered criteria are:

1. Agent interaction: in some models agents need to interact with one another. These interactions may be limited to the agent perception field or not. In a parallel context, interactions between agents are more difficult to implement than in a centralized context as agents cannot always directly access to the memory representation of other agents. For this reason this criterion expresses the support given by the platform regarding communications between agents.
2. Communication layer: most of the platforms rely on already existing communication layers. The communication layer properties have an impact on the platform properties and thus on model implementation. Some communication layers as MPI are more HPC oriented while others as RMI are less efficient and more suited for less coupled architectures as networks of workstations.
3. Synchronization: in a multi-agent simulation run, the scheduler must guarantee that the event processing order respects the simulated time order. On parallel machines each node runs at its own speed, it is not physically synchronized with the others. This leads to different physical time while running the same time step. On the other hand, the number of agents in each platform instance may differ and may impact the time taken to run a time step. As a result the agent scheduler on different instances may be in different time steps if no synchronization is introduced, without having the possibility to synchronize on the physical time. According to [36] there are several ways to synchronize simulation instances. The conservative synchronization imposes each instance to be in the same time step, i.e. a synchronization step as a global barrier must be done on each node at the end of each time step. The optimistic synchronization relaxes the constraint of every node running in the same time step and uses a roll-back mechanism when causality errors are detected. The optimistic synchronization avoids blocking situations and thus provides possible performance gains compared to the conservative one. This gain could however be reduced, canceled or even inverted depending on the number of causality errors to be handled. Note also that the optimistic synchronization scheme only works if it is not always the same node that runs slower than the others. In this case, the global running time of the simulation depends on the running time of the slower node and is thus not impacted by this optimization.
4. Load balancing: to get good performances from a simulation run, all the processors allocated by the resource manager must be efficiently used, i.e. used as much as possible. In PDMAS the processors are mainly used to run agent behaviors. The load of a node then depends on the number of agents it has to run and on the behavior of the agents. When the load is not balanced between the nodes, some nodes are idle and waiting for the others, for instance on the time step synchronization. As a result the performance of the whole system is usually not very good. The load balance may be managed by the application or automatically by the PDMAS. The load balancing criterion refers to an automatic load balancing function in the platform.
5. Architecture: there are several possible platform architectures to distribute the processes on the processors and manage information exchanges between these processes. This criterion is set to *distributed* if all the processes are autonomous and participate to the simulation. It is set to *master/slave* if only one process starts the others to run the simulation.
6. Scalability: this criterion refers to the possibility of running large simulations, in terms of number of used processors or cores. This is an important criterion as it measures the ability of the platform to scale with the number of simulated agents.

5.2. *Analysis*

Table 1 gives a synthetic representation of the comparison for the agent development and the properties of the agent entity in the platform. Most platforms use classical languages such

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001

Table 1 – Agent development and the properties of the agent entity in the platforms.

Platform	Programming language	Agents representation	Agents state	Agents behavior	Agent identification
RepastHPC	C++/RLogo	Objects	Variables	Methods	Unique ID
D-MASON	Java	Objects	Variables	Methods	Unique ID
Flame	XMML/C	CSXM/C [37,22,38]	Acyclic State Machine [22]	Transition/State/Functions [38,22]	Unique ID
Pandora	C/C++	Objects	Variables	Methods	Unique ID
JADE	Java	Objects	Variables	Methods	Unique ID
PDES-MAS	C/C++	Objects/Agent Logical Process (ALP)	Variables	Methods	Unique ID
Ecolab GraphCode	C++	Objects	Variables	Methods	Unique ID
SWAGES	Poplog/Java [32]	Objects	Variables	Methods	Unique ID
MACE3J	Java	Objects	Variables	Methods	Unique ID
ABM++	C/C++	Objects	Variables	Methods	Unique ID

as C–C++ or Java to define agents, except the Flame platform which uses the XMML language. The XMML language is an extension of the XML language designed to define X-Machines. Note that the RepastHPC platform implements, in addition to the C++ programming language, the widespread Logo agent language. The Repast-Logo or R-Logo is the Repast implementation of Logo for C++. It allows to simplify the simulation implementation at the price of a lower power of expression compared to C++.

Agents are usually defined as objects with methods representing behaviors and variables representing states. All agents are identifiable in the simulation using a unique identification. An agent container gathers all the agents. This container is cut and distributed in the case of parallel execution. The agent implementation is different for the Flame platform that does not use the object concept to define an agent but rather uses automata called X-Machines and more precisely Communicating X-Machine [37]. In a X-Machine, a behavior is represented by a state in the automaton and the execution order between behaviors is represented by transitions. This difference changes the programming logic of a model but induces no limitation compared to other platforms because agents are encoded in the C language.

Table 2 groups criteria about the general properties of the platforms regarding the simulation and the guaranties provided. While most platforms implement a time model to animate the simulation time discretization, the JADE platform rather targets the support of agent based applications on distributed resources. It thus does not need to provide any time nor synchronization support and it is flagged Not APplicable (NAP) for this criterion in the table. On the other hand recent works have proposed a control framework for time-dependent multi-agent systems [39] on JADE which prove that this platform can be completed to manage time.

For the agent synchronization, event or time driven, all platforms use the time-driven approach except RepastHPC and PDES-MAS which are based on the event-driven approach and JADE that does not provide time nor synchronization support as previously explained. RepastHPC also allows to fix a periodicity to each scheduled event, so that we can easily produce time-driven simulations.

In parallel platforms as PDMAS several runs are done simultaneously, note that only Flame does not support simultaneous event executions.

Simulating a model must be deterministic, that is to say, different runs with the same set of parameters should always produce the same results. Most platforms, take reproducibility into account. Platforms for which we did not find any information on the reproducibility property are marked as Not AVailable (NAV) for this property. Note however that the reproducibility concern is complex in a parallel context (see for instance the issue on random numbers generation in [35]) and the property given in the table only reflects what is given in the papers.

The most used random number generator is Mersenne Twister. For different platforms, we did not find any data about pseudo random number generation. They are marked as Not Available (NAV) for this property.

Table 3 summarizes the criteria of the platform characteristics regarding parallelism. Globally we can note that all studied platforms meet the demands for the development of parallel simulations.

All platforms allow agents to communicate either *internally* with agents on the same node, or *externally*, with copy of agents on different nodes. The D-MASON and Pandora platforms propose remote agent method invocations to communicate with other agents while the other platforms use messages to communicate between agents.

The communication layers of most platforms are based on MPI. This is not surprising for platforms targeting HPC systems as this library is mainly used on these computers. Note that the D-MASON platform is based on the JMS communication service despite it is not the most scalable solution for distributed environments. An MPI version of D-MASON is in development. Finally, the Jade platform communication support is based on transport protocols as Java-RMI, HHTP or IIOP that do not fit parallel applications as they are based on synchronous calls and cannot make an efficient use of high performance networks (e.g. InfiniBand).

To efficiency exploit the power of parallel resources the computing load must be balanced among the nodes. There are different ways to balance the computing load. It can be balanced at the beginning of the simulation (Static) or adapted during the execution (Dynamic). A dynamic load balancing is usually better as it provides a better adaptation in case of load variation during the model execution, but it can also be subject to instability. Most platforms use dynamic load

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001

Table 2 – Properties of the platforms regarding the simulation and the guaranties provided.

Platform	Agent synchronization	Simultaneous events	Reproducibility	Random numbers
RepastHPC	Event-driven	Yes [22]	Yes	Mersenne Twister [40]
D-MASON	Time-driven	Yes [22,41]	Yes	Mersenne Twister
Flame	Time-driven	No [22]	NAV	Marsaglia [42]
Pandora	Time-driven	Yes	Yes [27]	NAV
JADE	NAP	NAP	NAV	NAV
PDES-MAS	Event-driven	Yes	Yes	NAV
Ecolab GraphCode	Time-driven	Yes [22,33]	No	UNU.RAN [43]
SWAGES	Time-driven	Yes	NAV	Mersenne Twister
MACE3J	Time-driven	Yes	Yes [34]	Own random generator
ABM++	Time-driven	Yes	NAV	NAV

Table 3 – Platform characteristics regarding parallelism.

Platform	Agent interaction	Communication library	Synchronization	Load balancing	Architecture	Scalability
RepastHPC	Local/Remote (Copy of others) [29]	MPI/Boost lib	Conservative	Dynamic	Distributed	1028 proc.
D-MASON	Local/Remote (AOI) [41]	JMS ActiveMQ/MPI	Conservative	Dynamic	Master/Slave [41]	36 proc.
Flame	Local/Remote (Agent-agent+Sphere of influence) [22]	MPI	Conservative	Static [22]	Distributed	432 proc. [38, 22]
Pandora	Local/Remote [27]	MPI	Conservative	Dynamic	Distributed [27]	NA
JADE	Local/Remote	RMI/IIOP/HHTP/JICP	NA	NA	Distributed [26]	NA
PDES-MAS	Local/Remote (Communication Logical Process and Sphere of influence)	MPI	Optimistic	Dynamic	Distributed	NA
Ecolab GraphCode	Local/Remote (Copy of others)	MPI	Conservative	Dynamic	Distributed	NA
SWAGES	Local/Remote	SSML	NA	NA	Distributed	50 proc.
MACE3J	Local/Remote (MACE3J Messaging System, MMS)	(Objects/Interfaces)	Conservative	NA	Distributed	48 proc.
ABM++	Local/Remote (Copy of others)	MPI	Conservative	NA	Distributed	NA

balancing except the Jade and Flame platforms. In [44] the authors propose a way to use dynamic load balancing with the Flame platform.

Last, several architectures could be used to distribute the processes on the processors. The most used architecture is distributed, all the processes are autonomous. Only D-MASON uses a Master/Slave architecture. This choice can be explained as it better fits with the ActiveMQ JMS library as a communication library. Note that in the latest version of D-MASON, the communication schema is based on MPI. In this case the architecture is distributed.

Finally, note that we did not find any information on the scalability property for several platforms. They are marked as Not Available (NA) for this property.

For each platform we tried to download the source or executable code and we tried to compile it and test it with the provided examples and templates. Some of the platforms cannot be further included in our study because there is no available source code or downloadable executable (MACE3J [34], JAMES [45], SWAGES [32]), or because only a demonstration version is available (PDES-MAS [30,31]), or because there is a real lack of documentation (Ecolab [33]). It was not possible to develop a model for these platforms and thus to assess their parallel characteristics and performance. These platforms are then not considered in the remaining of the paper.

From this first assessment we can note that some platforms have rather been designed to support multi-agent simulations, i.e. animating a model composed of numerous entities (agents) that implement an individual behavior, whereas others are more focused on supporting the distribution of agent based applications, i.e. applications based on components (agents) that provide a service like equipment monitoring or interoperability management. Hence, during the model implementation we noted that the Jade platform seems to be more oriented for equipment monitoring rather than to implement model simulations and run them on a large platforms. The Jade web site presents it as a base for "*peer-to-peer agent based applications*". It is indeed not HPC compliant [46] as it cannot be run in batch mode. Note that implementing a medium size model above JADE is however possible as shown in [47]. Jade is then not included in the rest of the comparison that concentrates on the D-MASON, Flame, Pandora and Repast-HPC platforms.

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001

6. Analysis of distribution support

Developing a parallel multi-agent simulation is not an easy job and taking advantage of the platform support may greatly ease it. Our objective here is to assess the support proposed by the platforms for running large simulations. For this reason we concentrate on the distribution characteristics of the platforms to define evaluation criterion. To better understand the way these characteristics may impact a simulation development and to validate that they are operational, we also assess the development of a model on the four remaining platforms.

6.1. Method

The evaluated criteria mainly focus on the distribution properties of the platforms. When implementing a model to distribute a simulation we face four main issues:

1. Distribution: this criterion refers to how agents are distributed on the processors or nodes. As we run in parallel the global set of agents must be split between the processors to benefit from the whole processing power. The agent distribution is done by the platform but this distribution greatly influences the performance so it is important to notice how this distribution is done.
2. Communication: this criterion refers to the implementation of the communication layer of the platform. As for all parallel applications an efficient communication scheme is needed for the performance and the scalability of the simulations.
3. Coherency/synchronization: when agents are run in parallel we need to manage concurrent accesses to common data. The data coherency or access synchronization and their properties are a necessary knowledge when implementing a model as they define the limits of the platform support. If no synchronization is provided then concurrent data accesses should be avoided in the model, or managed by the model developer.
4. Load balancing: this criterion refers to an automatic load balancing function provided by the platform. This is the dynamic part of the agent distribution as it re-distributes agents during the simulation run. This implies that platform is able to move agents from one node to another during the simulation run. This is usually referred to as agent migration. This implies for the platform to move the agent data between the nodes and to be able to redirect communications. As for distribution, load balancing and its implementation is thus an important criterion to consider.

These properties are important to understand how to efficiently implement models and what are the constraints imposed to the model design. In addition, these criteria are a way to show the scope of each platforms regarding their distribution support. Truly and deeply understanding and assessing these properties can however not be done by just reading the research papers and documentations. For this reason, we have implemented the same reference model on each of the four platforms. By implementing this model we did one step further in the understanding of the platform use and of their properties. We have also been able to check how far their claims are true. As we were able to implement our model for the four platforms, we can then consider that they truly offer a functioning parallel multi-agent support.

6.2. Reference model

Agent based simulations are usually designed with an environment that defines position coordinates or contextual data [48]. The simulator iteratively applies the agent behaviors on their own data and on the environment data. Note that, although some agent based simulations, as social network models, do not include an environment, the reference model has to use situated agents as three out of the four assessed platforms base their distribution facility on the agent coordinates in the environment. On the other hand this choice may limit the conclusions of the paper to models tied to a grid based environment.

The proposed reference model is based on three important behaviors for each agent: perception, communication between agents and/or with the environment and mobility. We chose these behaviors as they are representative of general behaviors found in multi-agent models and match the agent definition given by Ferber (see Section 2).

Fig. 1 gives an AML [49] (Agent Modeling Language) representation of our reference model.[1] The *Environment* is represented by a square grid. *Agents* (*Person*) are mobile and move randomly. A *vision* characterized by the "radius" property is also associated with each agent. It represents the limited perception of the agent on the environment.

Each agent behavior is composed of 3 sub-behaviors that are executed at each time step in the following order:

1. The *walk* behavior allows agents to move in a random direction on the environment, in one of its 8 Moore neighbor cells in the grid (or less if the agent is close to a border). This behavior is used to test the mobility and the perception of the agents. As the agents walk through the environment they discover other agents and other parts of the environment. Interactions and communications with the environment are tested with this behavior.
2. The *interact* behavior allows agents to interact. The agent sends a message to all the agents in their perception field. The message is only composed of the agent id and an integer value. This behavior intends to simulate communications between agents and to assess the communication support of the platforms.
3. The *compute* behavior simulates the workload generated by running the agent inner algorithms. This behavior computes a one dimensional forward "Discrete Fourier Transform (DFT)" [50] on a table.

Some of the settings of the reference model are common to all the experiments presented in Section 7. The size of the grid environment is 2000 × 2000. The perception radius for the agents is set to 3. All the random laws (choice of neighbor cell and the initialization of the DFT table) used are uniform laws. The size of the DFT table is set to 128.

Using this reference model and implementing it on the four platforms we get a better understanding of their properties. In the following, we synthesis the properties of each platform regarding the four defined criteria for distribution.

[1] The implementations of the model on the four assessed platforms are available at http://dx.doi.org/10.6084/m9.figshare.1562356.

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001

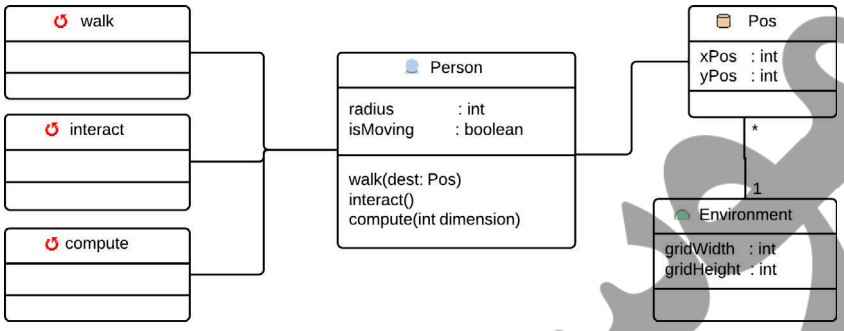
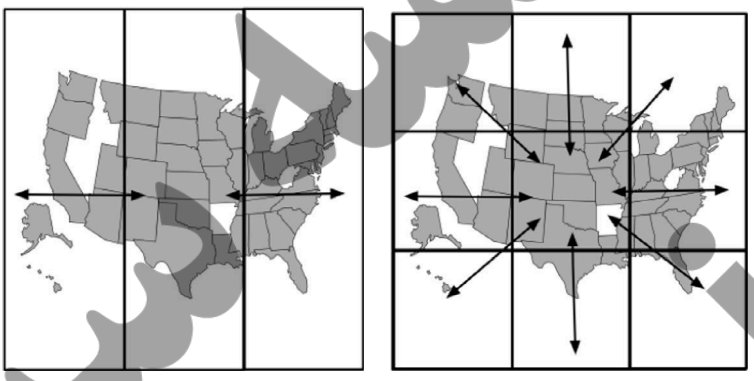


Fig. 1 – AML representation of the reference agent model.



(a) Y distribution. **(b) XY distribution.**

Fig. 2 – Grid distribution in the D-MASON platform.

6.3. *Distribution*

Distribution is the basis of a parallel model. The way the distribution is implemented directly affects, on the one hand, the model development complexity and, on the other hand, the execution performance.

6.3.1. *D-MASON*

From the distribution view point, D-MASON is a generic platform as agents could have or could not have a position in a Cartesian space representing the environment. The environment is divided into cells, or partitions, that are assigned to processors to distribute the simulation. D-MASON proposes three ways to distribute a simulation over several cores. Two ways are based on field partitioning or grid partitioning and the last one is based on network partitioning. Overlapping areas (Area of Interest or AOI) are defined to guarantee the continuity of agent perception through node borders.

Field or Grid partitioning. Fig. 2 represents the two distribution mechanisms available in D-MASON for grid partitioning. The "Y Distribution" consists in dividing the environment in horizontal cells on the Y axis as shown in Fig. 2(a). The "XY Distribution" consists in dividing the environment in square cells using the X and Y axes as shown in Fig. 2(b).

The grid environment is distributed by cutting its cells and mapping them on the nodes of the execution platform. To provide a continuous environment, D-MASON uses overlapping zones, also called "Area of Interest" (AOI) as shown in Fig. 3. This mechanism consists in locally copying a part of each neighbor cell or adjacent partition. The overlapping zone offers the opportunity for agents to keep a global perception field even when their environment is cut between several nodes. To maintain the continuity each cell exchanges information with its direct neighbors before each time step.

Fig. 3 – Example of Area of Interest in a Y distribution.

During the distribution phase only the environment is considered for dividing the simulation. Agents are not taken

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001

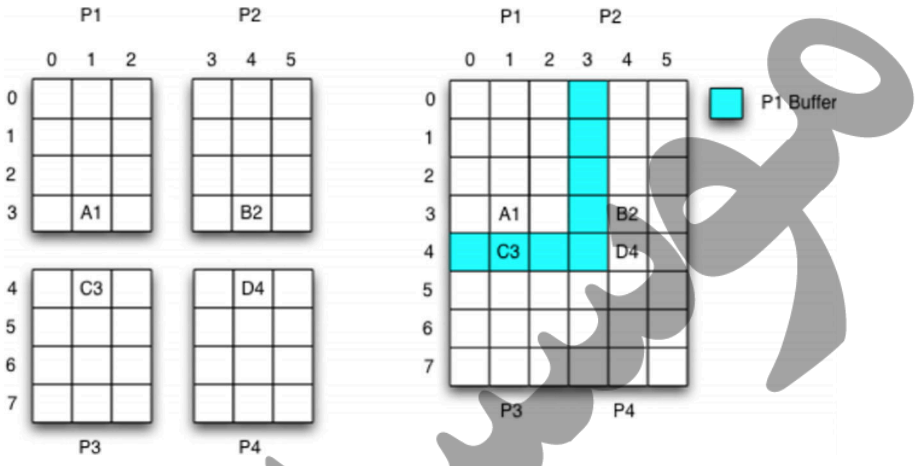


Fig. 4 – Schema of a grid projection on 4 processors in RepastHPC [29].

into account for this phase. Agents density is however taken into account for the load balancing phase.

Network partitioning. Network field is a way to represent a graph structure in the simulation. To distribute a graph over several processors, frameworks like ParMetis [51] or Metis are used. ParMetis is an MPI based parallel framework that implements algorithms for partitioning unstructured graphs, meshes and so on. ParMetis extends the functionality of Metis to distributed and large scaled graphs. We did not find more information about the network partitioning mechanism which was added in a recent update of the platform.

It is important to note that, in the D-MASON platform, we can use different types of layer in the same simulation. In other terms, a spatial distribution could be used for the environment while a network partitioning is used to represent interaction between agents. In that case, the distribution is only based on the field partitioning.

To distribute agent simulations, RepastHPC uses a mechanism called "Projection", adapted from the Repast S platform. It represents the environment in which the agents evolve. Projections can be of two types, either grid or network. The projections are used to impose a structure to the agents in which they can evolve. Overlapping areas (or area of recovery) are defined to manage the continuity of agent perception through node borders.

6.3.2. *RepastHPC*

Distribution in RepastHPC as several similarities with the D-MASON proposition: grid projections are roughly equivalent to grid partitioning and a network projection is equivalent to a network partitioning.

Projection Grid. The Grid Projection represents agents in a Cartesian space. To distribute a model over several processors, the environment is divided into cells of equal size. These cells are regularly distributed on the processors. Each processor is responsible for a sub-part of the grid. The sub-parts of the grid are connected with overlapping areas.

Fig. 4 represents a distribution schema of a grid projection on 4 processors. The grid ranges from coordinate (0,0) to coordinate (5,7). Processor P1 is responsible for the sub-part (0.0) × (2.3), P2 is responsible for the portion (3.0) × (5.3) and so on. In our example, the size of the overlapping areas is set to 1. In this case, P1 contains an area buffer that includes the entire column 3 of processor P2 and line 4 of processor P3.

Network Projection. Network Projection [28] is a way to represent a graph structure. Fig. 5 represents a diagram of a network projection with two agents distributed on 2 processors.

In order to divide the graph on several processors while maintaining the links between vertices dispatched on different processors, a copy of neighbor edges and vertices is made. Unfortunately, no information is available in the RepastHPC documentation on how the graph is distributed on multiple processors so that this projection is hardly usable.

As for D-MASON it is possible to use several projections for the same model. For instance, a Grid projection for a geographical environment and a Network projection for agent interactions.

6.3.3. *Flame*

Agent distribution in Flame simulation differs from the two previous distribution as it is performed statically [52]. It is defined or computed at the beginning of the simulation and does not change during the execution. Distribution choices are based on the graph of communications between agents and aim at optimizing the overhead generated by inter-processor communications. Two methods can be used to distribute a simulation in the Flame platform: Geometric (or Separator) and Round Robin. The distribution method is set at the beginning of the simulation using a parameter.

Geometric distribution. This distribution spreads agents over the processors on the basis of their position in a Cartesian space. The partitioning is based on their coordinates that could be either in 2D or 3D. Note that in this case the environment is not cut on a grid basis, it is still considered as continuous and the partitioning is only driven by their distribution. The goal of this distribution is to group the nearest agents as they can potentially communicate together and generate a lot of communications.

Round Robin distribution. If the agents are not located in a Cartesian space, Flame provides a distribution mechanism for

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001

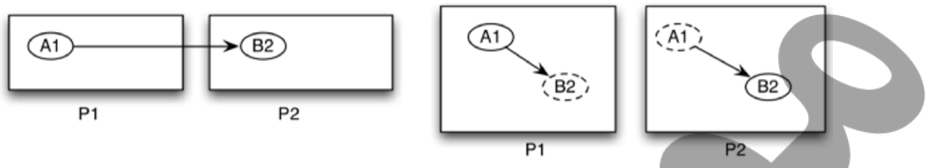
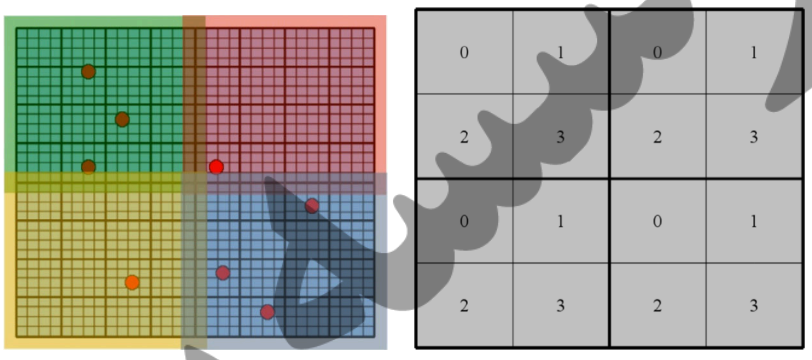


Fig. 5 – Diagram of network projection in RepastHPC [29].



(a) Schema of distribution. **(b) Execution schema.**

Fig. 6 – Distribution and execution order in Pandora [53,54].

agents with a Round Robin method: agents are assigned one by one to each processor in a round. The method does not take the agent behaviors into account but it is possible to add a discriminator, such as the agent type, to influence the distribution. For instance agents of a same type can be grouped in one or more partitions if these agents communicate more with agents of their type rather than with another agent type.

6.3.4. *Pandora*

In the Pandora platform distribution is made by dividing the environment in several portions like in Grid partitioning of the D-MASON or RepastHPC platforms. Fig. 6(a) comes from a presentation of the Pandora platform. It shows that the distribution is made by dividing the environment and that overlap zones are used to keep the partial continuity of the environment even when they are distributed over several processors.

Note that, except FLAME, all the platforms use a geographical environment to dynamically distribute the agents. The simulations and their results will be thus tied to this constraint. On the other hand FLAME only provides static distribution of the agents from the initial configuration and does not balance the load of the nodes during the runs.

6.3.5. *Communication*

Communication is a key functionality in the platform interface for parallel models. It is basically proposed as either remote invocation on agents or asynchronous messages.

6.3.6. *D-MASON*

Communication between agents using messages is not implemented. Communications between agents are carried out by method calls on the target agents. This functionality is only available for agents executed on the same processor or in the overlapping zones. If the target agent is not running on the same processor nor in the AOI, it is not possible to communicate with it using a grid field environment. Nevertheless, using a network field can overcome this limitation. In fact, network field allows any agent to communicate with another agent when a link or an edge exists between them.

In addition, D-MASON proposes a way to use global parameters over several processors. This can be a useful support to spread information to all the agents of a simulation.

6.3.7. *RepastHPC*

Communication using messages is not implemented. Communications between agents are carried out by method calls on the target agents but are only available for agents called on the same processor. If the called agent does not run on the same processor, the caller agent can request a copy. By this way, calling methods on remote agents are possible. On the other hand if an agent copy is modified, the changes are not reported to the original agent. Updating a copy agent from the original agent is however possible to keep information coherency.

6.3.8. *Flame*

In Flame an agent cannot send a message directly to another agent. Communications are implemented using “message boards” or message tables. Each processor owns a message board that is divided between its agents. Agents can read (receive) or write (send) messages on their message board part using the message board API. Agents can send or receive messages of the different types defined in the XMML file of the simulation. An agent cannot send and receive messages in the same behavior or state. It has to be implemented on

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001

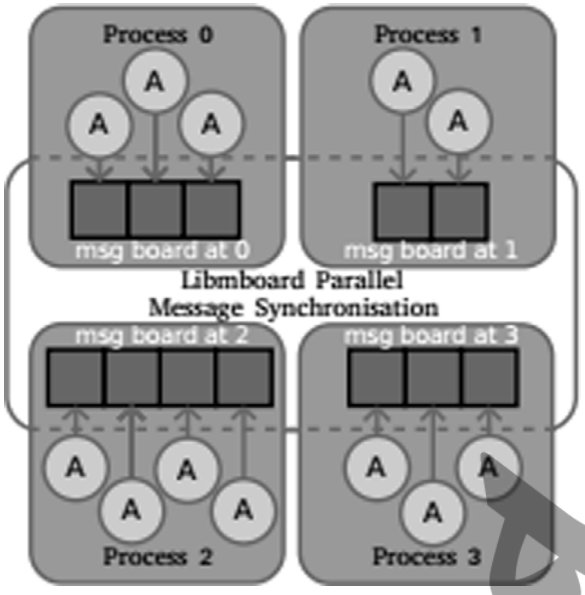


Fig. 7 – Distribution schema of MessageBoards in Flame [52].

two separate states. Parallelization of a simulation is based on the distribution of the message boards as shown in Fig. 7.

As the agents can run on different processors (depending on the distribution), Flame uses broadcasting as a communication method instead of trying to locate them. It is however possible to limit the number of recipients per message through filters. In this case the message is simply broadcasted to a group of agents.

The message board library, or libmBoard, is required to manage the distribution, the creation, the deletion, the synchronization and the access to messages in MessageBoards. The libmBoard library uses MPI to communicate between processors and uses POSIX threads (pthreads) to handle data management and inter-process communications.

6.3.9. *Pandora*

Communications are performed using MPI and more precisely using μsik library [55]. The μsik library is designed to manage communications between nodes and it is specifically designed to execute discrete-event simulations on parallel resources. The Pandora platform automatically generated the code to avoid the user from writing MPI code. Nevertheless, the Pandora platform limits inter-agent communications: agents can only communicate within there perception field (communication must be local) [53].

6.4. *Coherency/Synchronization*

The coherency/synchronization property real differs between the four platforms: each solving the coherency issue in its own way to provide as much transparency as possible to the developers.

6.4.1. *D-MASON*

D-MASON platform implements a conservative synchronization that guarantees against causality errors. To achieve this conservative synchronization, each timestep of the simulation is divided into 2 steps: (1) Communication/Synchronization and (2) Simulation run. There is a synchronization barrier at the end of each timestep. Agents in cell c do not run timestep i until their neighbor cells have not finished with their simulation timestep $i-1$. Then each cell sends information to its neighbors about agents that are in the AOI or that will migrate. For timestep i the cell c behavior is thus computed based on the data of step $i-1$ of its neighbor cells.

In the D-MASON platform, the different communication strategies for synchronization are based on ActiveMQ and MPI [56]. Early versions of D-MASON use Active MQ JMS with the "Publisher/Subscriber" paradigm. This paradigm is however not natively supported by MPI. Authors have then implemented it over MPI. In the D-MASON platform, MPI processes are assigned to D-MASON Workers and they use "communication groups" which describe the communications through the cells.

The synchronization model uses and respects five assertions:

- each cell creates its own topic to publish its updates,
- each cell subscribes to at least one topic (at least one neighbor cell). A cell cannot start a new simulation step until it receives updates from all the topics it has subscribed to,
- subscription is static: each cell executes the subscription routines only before starting the simulation,
- the publish on a topic where multiple fields (i.e. other cells) are subscribed can only happen when all the updates of each field are available,
- the publish is invoked only at the end of a simulation step.

With the five above assertions, there are three kinds of communication strategies for synchronization depending on the MPI routine used: BCast that uses broadcasting, Gather that uses gathering of the messages, and Parallel that uses send and receive in round to ensure a communication parallelism.

6.4.2. *RepastHPC*

Synchronization between processors in the RepastHPC platform is performed in four cases according to [29]. First, when a processor needs one or several copies of agents present on another processor. This synchronization is required to keep the simulation in a consistent state. Second, when a processor has non-local agent or edge copies from another processor, then the copies must be updated to the latest information from the original object. Third, the overlapping areas of a grid should be updated at each time step. And last, when an agent must be fully migrated from one processor to another. For example, when the agent moves into the environment.

Most synchronization mechanisms are not in charge of the programmer. Programmers only have to write pattern package that defines the necessary information to be synchronized in the agents. The pattern package consists of two methods *Provider*, used to serialize the data before sending, and *Receiver*, used to de-serialize the data when receiving. These two methods are a way to serialize agent information in order to synchronize the data needed when migrating agent from a processor to another.

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001

6.4.3. *Flame*

Synchronization between processors in the Flame platform is conservative and is performed through the MessageBoards. Actually, all the exchanges between agents are performed using messages so that the synchronization relies on the MessageBoard synchronization which, in turn, relies on broadcasting messages to the processors participating to the simulation. In this way, all processors have a unified view of the simulation. The synchronization of the MessageBoard is performed in two steps by first asking for synchronization and then performing it. In the first step, when a processor has finished to execute its agents, it locks its MessageBoards and sends a request for synchronization in a queue. After this step, it is still possible to make actions which do not need the MessageBoard. When all processors have locked their MessageBoards, the next step is performed. In the second synchronization step messages are exchanged between MessageBoards. The synchronization phase is a blocking phase. After these two phases, MessageBoards are open and the simulation continues.

6.4.4. *Pandora*

Synchronization in the Pandora platform is conservative [27]. Like in the RepastHPC and D-MASON platforms, data and agents located in the overlapping areas are copied and sent to the neighbors every time step in order to keep data up-to-date. The size of the overlapping zone is defined as the maximum size of the perception field of any agent [57,54]. To solve the synchronization problem of overlapping zones between processors – solve conflicts between agents on different processors – the Pandora platform uses a different method than the other platforms. Each portion of the environment which is distributed over several processors is also distributed in four equal sub-portions, numbered from 0 to 3, see Fig. 6(b). During the execution phase of agents all the processors execute sequentially each of the sub-portions. In other terms, all processors execute the sub-portion 0, then, the sub-portion 1 and so on. In this way, there is no possibility of conflicts because all sub-portions are not adjacent. Once one portion is finished, the modifications of the overlapping zones are sent to the neighbors. When all the sub-portions are executed the entire state of the simulation is serialized and a new timestep can be evaluated.

6.4.5. *Load balancing*

We present here the information we have found concerning load balancing in the tested platforms.

6.4.6. *D-MASON*

Load balancing in the D-MASON platform is performed dynamically [58]. The proposed method can be used in 2D and 3D environments. Load balancing is implemented as an additional step in the timestep, in addition to synchronization and simulation. At the end of each timestep, a density in terms of agents is computed on each cell of the environment. Agents of high density nodes are dispatched on neighbor nodes to balance the load.

6.4.7. *Flame*

There is no dynamic load balancing in the Flame platform. The load balancing is done statically at the beginning of the simulation with the proposed distributions. Note that Marquez et al. in [44] propose a load balancing schema for the Flame platform but it is not yet implemented in the main release.

6.4.8. *RepastHPC*

We did not find any information about mechanisms implemented in the RepastHPC platforms to perform the dynamic load balancing so that this property is not tackled here.

6.4.9. *Pandora*

Although the Pandora platform is used in different projects like SimulPast [59], there is a lack of explicit information. The implementation of the platform and of the mechanisms it uses are not accurately documented. In particular we did not find information on the way the load balancing is realized.

6.5. *Synthesis of the parallel properties*

Table 4 summarizes the properties of the four assessed platforms regarding the parallelism support.

7. Performance evaluation

The objective of parallelizing a model is either to get better running performance or to run larger, or more accurate, models that are too big to fit in the memory of one machine. The memory size issue is addressed by just adding more machines: their memory is added to the global amount. The performance issue is addressed by giving more cores to the simulation. To prove the efficiency of PDMAS in addressing these two problems experiments must be conducted to measure the performance obtained when running larger models on more cores. We have then run our reference model on the four previously selected PDMAS. We detail in this section their performance results and our observations when running them on HPC resources, i.e. HPC clusters.

7.1. *Experimental settings*

For the performance evaluation we have implemented the reference model defined in Section 6.2 on the four functional platforms: RepastHPC, D-MASON, Flame, Pandora. During this model implementation, we did not encounter noticeable difficulties. Note that, thanks to the PDMAS support, the model does not change whatever the size of the execution platforms.

We have run the reference model on a 1280 core cluster using the SGE batch system. Each node of the cluster is a bi-processors, with Xeon E5 (8*2 cores) processors running at 2.6 Ghz frequency and with 32 Go of memory. The nodes are connected through a non blocking DDR infinyBand network organized in a fat tree. The system, and in particular the network, is shared with other users. The batch system guaranties that each process runs on its own core without

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001

Table 4 – Comparison of synchronization between platforms.

	D-MASON	RepastHPC	Pandora	Flame
Communication strategy	Publisher/Subscriber	Distributed	Distributed	Distributed
Mechanisms	Overlapping zones	Overlapping zones	Overlapping zones	MessageBoards
Overlapping configuration	Yes	Yes	No	No
Paradigm	Master/Workers	Distributed	Distributed	Distributed
Agent migration	Yes	Yes	Yes	No
Synchronization based on	$t-1$	$t-1$	t	$t-1$
Synchronization management made by	Workers	Process	Process	MessageBoards
Potential Communication Model	$n-n$	$n-n$	$n-n$	$n-n$
Communications routines	MPI_Bcast, MPI_Gather	NA	μsik library	MB_SyncStart, MB_SyncEnd

sharing it with other processes (no time sharing). Note that this is a common configuration for assessing the performance of a parallel execution as HPC clusters in computing centers are usually shared between users.

When submitting a job the number of requested cores is given to the scheduler with a Robin option. This option means that the scheduler tries to dispatch as much as possible the processes on different nodes. These experimental settings may generate small variations in the running time so that we computed these variations on the performance results. The observed variation is low, less than 0.21% (obtained with Flame on 128 cores). For that reason, each measure is taken 10 times and we thus just report the mean of these measures (box plots would not give much more information).

7.2. *Performance results*

Although we have been able to run the four platforms, D-MASON, Flame, Pandora, RepastHPC, on a standard workstation, only three of them (RepastHPC, Flame and D-MASON) have successfully run on our HPC system. The Pandora simulations have deadlock problems even if we use the examples that are given with the platform. We were thus not able to include results in our performance comparison.

When analyzing the execution of the D-MASON simulations we have noticed that the platform uses multiple threads per processes and that we cannot control the number of threads during the simulation runs. Batch schedulers as SGE or SLURM, and more generally batch schedulers used in computing centres, however assume that only one thread is run per allocated core. Otherwise the additional threads may use cores that are allocated to other jobs. D-MASON thus does not respect the HPC constraints and it cannot be compared with the RepastHPC and Flame platforms. It would compare single threaded executions with multi-threaded executions which is not correct. On the other hand we have noticed that D-MASON generates good performance. It will probably become an interesting PDMAS platform once the platform reaches the mandatory maturity level (there are still some bugs and some constraints on the load balancing could be improved). For this reason the presented results only consider the Flame and RepastHPC platforms.

We have realized several executions in order to exhibit the platform behaviors concerning scalability (Fig. 8), workload (Fig. 9) and memory consumption (Fig. 10). To assess scalability we vary the number of nodes used to execute the simulations while we fix the number of agents. For workload we fix the number of nodes to 16, 32, 64 and 128 and we vary the number of agents in the simulation.

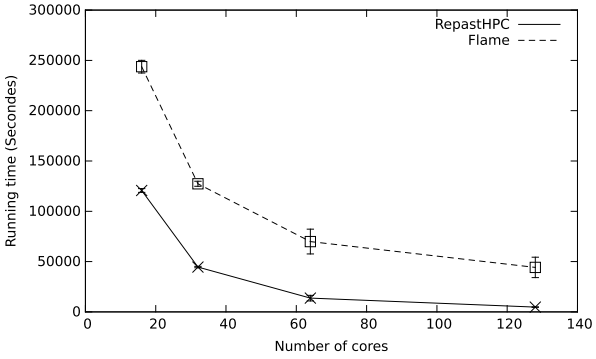
Scalability results for a model with 200.000 and 400.000 agents are given in Fig. 8. Note that the first execution is realized on 16 cores. We choose 16 cores because actually, most of workstations use processors with 8 cores so that these results may also be applied to a powerful bi-processor desktop. We can note that both platforms scale well up to 64 cores but the performance does not progress so well when more cores are used. RepastHPC results are better than Flame platform: for 16 cores, RepastHPC is about 2,02 time better than Flame whereas for 128 cores the difference is about 9,26 times better. This difference could be explained by the strategy used in the platform for agent communication. As the reference model is a communicating model and as Flame uses broadcasts to synchronize informations and communicate, it probably spends a lot of time in handling messages. In addition, we can note that, for 400.000 agents, the Flame platform scales well until 64 cores but more cores lead to a performance degradation. This under-performance could also been explained by the communication implementation of Flame.

Fig. 9 presents the workload performance of the two platforms for 16, 32, 64 and 128 cores. Load increase is obtained by setting the inner load of agents (generated by the size of the DFT calculus) and by increasing the number of agents in the simulation. The size of the DFT calculus is here set to 128. Note that the plots do not show results for all the X-axis values. This is because the platforms cannot run with so much agents on this number of cores. We can see that Flame does not scale as well as RepastHPC as less points are plotted in the figure. For instance, for 128 cores, we have not been able to run simulations with more than 500.000 agents.

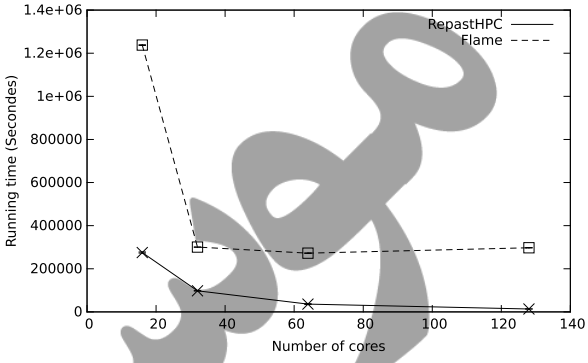
Fig. 9 also shows that RepastHPC really better reacts to load increase than Flame. Nevertheless, in Fig. 9 for 16 cores, we can note that the Flame platform offers better results than RepastHPC for 100.000 agents executed on the same number of cores. Obviously the used model does not use all the power of Flame as it is limited in terms of inter-agent communications. The question to answer is: is it due to the use of the X-Machines concept or due to the synchronization mechanisms? Another possible reason be the cost of the synchronizations provided by Flame when using remote agents and that it is not managed in RepastHPC.

Last, Fig. 10 represents the memory consumption of platforms executing a simulation over 16 or 128 cores. In Fig. 10, for 16 cores, we can notice that Flame uses about 2 times less memory that RepastHPC. On the opposite, for 128 cores, there

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001

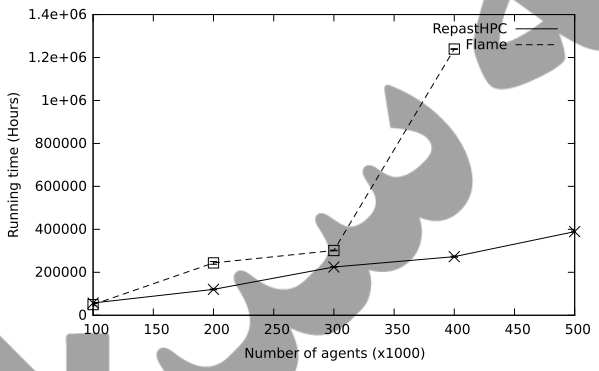


(a) Scalability for 200 000 agents.

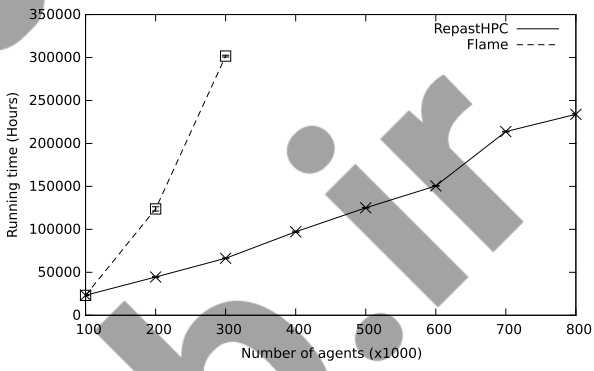


(b) Scalability for 400 000 agents.

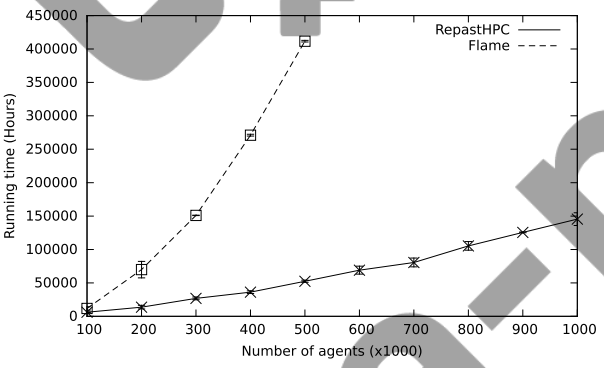
Fig. 8 – Platform scalability from 16 to 128 cores.



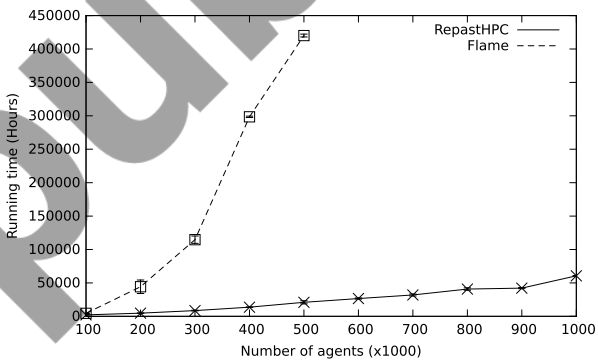
(a) Workload with 16 cores.



(b) Workload with 32 cores.



(c) Workload with 64 cores.



(d) Workload with 128 cores.

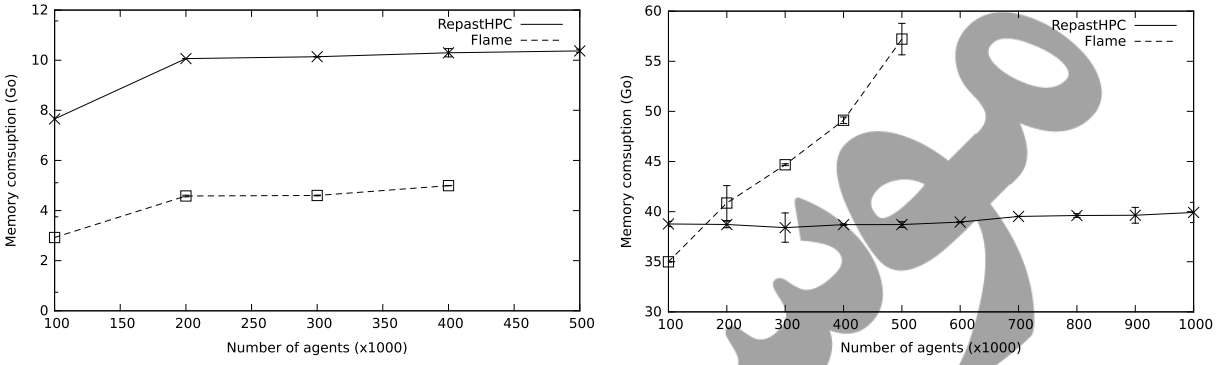
Fig. 9 – Platform workload for a DFT computation of 128 in each agent.

is a gap between the memory consumption of the two platforms. While RepastHPC stays approximately constant, Flame increases really faster. It uses more 57 Go of memory to run one simulation over 128 cores whereas RepastHPC uses about 39 Go of memory for the same simulation. These results could be explained by the inner implementation of agents in the platform and also the way to manage the communications and the synchronizations during the simulation. It is worth noting that only RepastHPC reaches 1 millions of agents without any problems and without consuming a lot of memory.

8. Synthesis

In this section we give our experience feedback over the tested platforms. This experience was gained during the bibliographical work, the discovery of the platforms, the implementation of different models and the experiments. It represents around half a year of accumulated work during the last two years and more than 250.000 h of computation. We summarize our impression and our informal opinion on

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001



(a) Memory 16.

(b) Memory 128.

Fig. 10 – Platforms memory consumption for 16 and 128 cores and DFT of 128.

the positive and negative points for each platform in the following.

8.1. *RepastHPC*

RepastHPC is a comprehensive platform which allows the use of different structures (Networks, Grids). These structures can be coupled in a same simulation to represent different types of interactions. Developing is facilitated by the number of tutorials which explain how to use RepastHPC. These tutorials are clear and detailed with a lot of steps. RepastHPC proposes two ways to implement multi-agent simulations: ReLogo and C++. Using ReLogo to develop a model is very simple but it does not benefit from some of the features of RepastHPC. On the other hand using C++ allows to implement more complex simulations with the drawbacks that good skills in C++ (notion of templates) are necessary.

There are however some limitations in the platform use as no communication is allowed between remote agents. This lack prevents from implementing some models, even simple ones, that need remote modifications of agents. On the other hand, RepastHPC will deliver high efficiency and scalability to read-only models.

After implementing our model on the Repast-HPC platform we have synthesized our feed back on this platform. The pro- and con-arguments are given in Table 5.

8.2. *Flame*

Flame is an interesting platform to develop parallel models as it uses a different approach compared to other platforms, i.e. X-Machines or state graphs. Although this approach implies to learn a new programming language it turns out that it is not more complex to implement a simulation on Flame than with the other platforms. Moreover a lot of examples are proposed that facilitate learning how to implement a simulation using XMML language. Using the X-Machine allows to hide the distribution issues to the developer and to transparently run simulations on parallel computers.

Unlike RepastHPC, the Flame platform allows to remotely modify agents through the use of messages and Message-Boards. This functionality is gained at the cost of less performance, even for read-only models. Two choices in the Flame design have lead to less performance, the absence of the overlapping area optimization and the systematic use of broadcast communications. Note that a new version of the Flame platform that solves the performance issue is announced [60] but not yet available at this time.

Table 6 summarizes the pro- and con-arguments for developing a model with the Flame platform.

8.3. *D-MASON*

Compared to RepastHPC and Flame, D-Mason is a more recent platform and its first implementation rather targeted network of workstations than HPC computers. It is however a fully featured platform, with a support for different ways of communication: ActiveMQ or MPI.

It is very easy to implement a simulation with D-MASON, all basic methods are already implemented. We do not need special skills or knowledge on Java to develop a model. In addition, the use of Maven simplifies the compilation of the simulations. D-MASON is a very great platform but it may lack maturity in the HPC cluster environment. As said previously we did not succeed in running a stable version of the MPI based platform with our reference model. Note that, as RepastHPC, D-MASON does not allow the modification of remote agents and is limited to read-only models.

D-MASON comes with a documentation that explains how to transform a centralized MASON model into a parallel D-MASON model. There is also a lot of example models that help the developer understanding the platform functionalities.

After implementing our model on the D-MASON platform we have synthesized our feed back with pro- and con-arguments shown in Table 7.

8.4. *Pandora*

As the previous platforms, Pandora is a full platform to implement parallel models. A very appreciable functionality provided by the platform is that the parallel code is automatically generated. We did not succeed in running a Pandora model on a cluster but we did it on a standalone desktop. Nevertheless, more documentation or tutorial could be appreciable instead of reading code examples

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001

Table 5 – Pro- and Con-arguments for the use of the Repast-HPC platform.

Pros	Cons
• Documentation, example and comprehensive tutorials • Ease of configuration and flexible scheduler: easily parameterizable even to perform complex scheduling • Does not need special knowledge in MPI • Large and very responsive community • Easy configuration and execution on clusters • Regularly updated	• Do not include all cases of communication between agents • Do not synchronize information with the original agent when a copy agent is modified

Table 6 – Pro- and con-arguments for developing with the Flame platform.

Pros	Cons
• Easy development once the XML syntax is known • Focus on the aspect state transition • Graphically play the simulation after execution with the Flame viewer (Optional but appreciated) • XMML Meta-model to validate the simulation • Automatic generation of the parallel code • Easy to switch between sequential and distributed executions	• Difficulty to validate and understand the XMML simulation model • Cannot send and receive messages in the same state • No point to point communication, all communications must go through the MessageBoard.

Table 7 – Synthesis of the use of the D-MASON platform.

Pros	Cons
• Simple and intuitive GUI • Facility of compilation: Maven • Responsive Community • Regularly updated • Propose two layers of communications (MPI or ActiveMQ JMS) • Propose three ways to synchronize using MPI (BCast, Gather and Parallel)	• Do not include all cases of communication between agents

to understand how to develop a simulation. Pandora also natively integrates GIS support, and the C++ simulation code is very simple to write.

Pandora does not provide remote communication but, thanks to its turning synchronization model, it solves the read–write model problem. This synchronization model however limits the model environment to 2D Grids and it does not support, as far as we understand, other environment types as networks.

Table 8 summarizes the pro- and con-arguments for developing a model with the Pandora platform.

9. Conclusion

In this article we have presented a comparison of different parallel multi-agent platforms. This comparison is performed at two levels, first at a qualitative level using criteria on the provided support, and second at a quantitative level, using a reference multi-agent model implementation and conducting performance evaluation. The qualitative analysis focuses on the ten platforms we found during our bibliographical search. A comparison shows the properties of the studied platforms regarding the development and implementation of agent entities, the guaranties provided and the parallelism support. We then concentrate on four platforms, the ones that are freely available and functional. For a more in-depth evaluation we have implemented a reference model which is a communicating model on each platform. With this implementation we assess the support provided by the platforms to develop parallel models and we concentrate on four properties: distribution, communication, synchronization and load balancing. In the quantitative evaluation we then assess the performance of the platforms for running the reference model on a HPC cluster. Only two platforms, namely RepastHPC and Flame, were able to run in this environment and respect the corresponding constraints. The comparison shows a noticeable difference in terms of scalability between the platforms. RepastHPC offers better performance results than Flame platform for the reference model.

When implementing our reference model we have noticed that the synchronization support of the platforms does not provide the same level of service: the RepastHPC and D-MASON platforms do not provide communication support for remote agents while Flame does it. Note that the platforms either provide overlapping areas, that optimize agent accesses to remote data, or message exchanges to modify remote agents. This support seems to be a key point in the platform performance. This difference is linked to

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001

Table 8 – Pro- and con-arguments for developing with the Pandora platform.

Pros	Cons
• Parallel and sequential code generation	• Poor documentation
• GIS support	• Do not include all cases of communication between agents
• Rapid prototyping	• Do not include all cases of synchronization between agents
• Easy to switch between sequential and distributed executions	
• Does not need special knowledge in MPI	
• Propose analysis tools (Cassandra)	

synchronization problems. For this reason, in our future work, we intend to better examine the efficiency of synchronization mechanisms in parallel multi-agent platforms. For example how to implement synchronizations during an execution and how to improve synchronization mechanisms in parallel multi-agent systems?

Acknowledgment

Computations have been performed on the supercomputer facilities of the Mésocentre de calcul de Franche-Comté.

REFERENCES

[1] R.M. Fujimoto, Parallel and Distribution Simulation Systems, first ed., John Wiley & Sons, Inc., New York, NY, USA, 1999.
[2] H.A. Simon, The Architecture of Complexity, Springer, 1991.
[3] K. Zia, K. Farrahi, A. Riener, A. Ferscha, An agent-based parallel geo-simulation of urban mobility during city-scale evacuation, Simulation 89 (10) (2013) 1184–1214.
[4] B. Bartley, Mobility impacts, reactions and opinions: traffic demand management options in Europe: the miro project, Traffic Eng. Control 36 (11) (1995) 596–602.
[5] J. Ferber, J.F. Perrot, Les systèmes multi-agents: vers une intelligence collective. InterEditions Paris, 1995.
[6] M. Woolridge, M.J. Wooldridge, Introduction to Multiagent Systems, John Wiley & Sons, Inc., New York, NY, USA, 2001.
[7] G. Carslaw, Agent based modelling in social psychology (Ph.D. thesis), University of Birmingham, 2013.
[8] V. Rodin, A. Benzinou, A. Guillaud, P. Ballet, F. Harrouet, J. Tisseau, J. Le Bihan, An immune oriented multi-agent system for biological image processing, Pattern Recognit. 37 (4) (2004) 631–645.
[9] M.J. North, N.T. Collier, J. Ozik, E.R. Tatara, C.M. Macal, M. Bragen, P. Sydelko, Complex adaptive systems modeling with repast simphony, Complex Adapt. Syst. Model. 1 (1) (2013) 1–26.
[10] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, MASON: A new multi-agent simulation toolkit, Simulation 81 (7) (2005) 517–527.
[11] S. Tisue, U. Wilensky, Netlogo: Design and implementation of a multi-agent modeling environment, in: Proceedings of Agent, Vol. 2004, 2004, pp. 7–9.
[12] E. Amouroux, T.Q. Chu, A. Boucher, A. Drogoul, Gama: an environment for implementing and running spatially explicit multi-agent simulations, in: Agent Computing and Multi-Agent Systems, vol. 5044, Springer, 2009, pp. 359–371.
[13] R. Tobias, C. Hofmann, Evaluation of free java-libraries for social-scientific agent based simulation, JASS 7 (1) (2004).
[14] R.H. Bordini, L. Braubach, M. Dastani, A. El Fallah-Seghrouchni, J.J. Gomez-Sanz, J. Leite, G.M. O'Hare, A. Pokahr, A. Ricci, A survey of programming languages and platforms for multi-agent systems, Informatica (Slovenia) 30 (1) (2006) 33–44.
[15] M. Berryman, Review of software platforms for agent based models. Tech. Rep., DTIC Document, 2008.
[16] B. Heath, R. Hill, F. Ciarallo, A survey of agent-based modeling practices (january 1998 to july 2008), JASSS 12 (4) (2009) 9.
[17] K. Kravari, N. Bassiliades, A survey of agent platforms, J. Artif. Soc. Soc. Simul. 18 (1) (2015) 11.
[18] S. Vialle, E. Dedu, C. Timsit, Parcel-5/parssap: A parallel programming model and library for easy development and fast execution of simulations of situated multi-agent systems, in: Proceedings of SNPD02 International Conference on Software Engineering Applied to Networking and Parallel/Distributed Computing, 2002.
[19] N. Bezirgiannis, Improving performance of simulation software using haskells concurrency & parallelism (Master's thesis), Utrecht University, 2013.
[20] F. Cicirelli, A. Giordano, L. Nigro, Efficient environment management for distributed simulation of large-scale situated multi-agent systems, Concurr. Comput.: Pract. Exper. 27 (3) (2015) 610–632.
[21] M. Lees, B. Logan, R. Minson, T. Oguara, G. Theodoropoulos, Modelling Environments for Distributed Simulation, Springer Berlin, Heidelberg, Berlin, Heidelberg, 2005, pp. 150–167.
[22] S. Coakley, M. Gheorghe, M. Holcombe, S. Chin, D. Worth, C. Greenough, Exploitation of hpc in the flame agent-based simulation framework, in: Proceedings of the 2012 IEEE 14th Int. Conf. on HPC and Communication & 2012 IEEE 9th Int. Conf. on Embedded Software and Systems, HPCC '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 538–545. http://dx.doi.org/10.1109/HPCC.2012.79.
[23] G. Cordasco, R.D. Chiara, A. Mancuso, D. Mazzeo, V. Scarano, C. Spagnuolo, A framework for distributing agent-based simulations, in: Euro-Par Workshops (1)'11, 2011, pp. 460–470.
[24] G. Cordasco, R.D. Chiara, A. Mancuso, D. Mazzeo, V. Scarano, C. Spagnuolo, Bringing together efficiency and effectiveness in distributed simulations: The experience with d-mason, Simulation (2013) 1236–1253.
[25] S. Coakley, R. Smallwood, M. Holcombe, Using x-machines as a formal basis for describing agents in agent-based modelling, Simul. Ser. 38 (2) (2006) 33.
[26] F. Bellifemine, A. Poggi, G. Rimassa, Jade–a fipa-compliant agent framework. in: Proceedings of PAAM, London, Vol. 99, 1999, p. 33.
[27] E.S. Angelotti, E.E. Scalabrin, B.C. Ávila, Pandora: a multi-agent system using paraconsistent logic, in: Computational Intelligence and Multimedia Applications, 2001, ICCIMA 2001, IEEE, 2001, pp. 352–356.
[28] N. Collier, M. North, Repast HPC: A Platform for Large-Scale Agent-Based Modeling, John Wiley & Sons, Inc., 2012, pp. 81–109. http://dx.doi.org/10.1002/9781118130506.ch5.
[29] N. Collier, Repast hpc manual, 2010. http://repast.sourceforge.net/docs/repast_hpc.pdf.
[30] T. Oguara, G. Theodoropoulos, B. Logan, M. Lees, C. Dan, PDES-MAS: A unifying framework for the distributed simulation of multi-agent systems. Tech. Rep. CSR-07-7, School of computer science research, University of Birmingham, 2007.

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001

[31] V. Suryanarayanan, G. Theodoropoulos, M. Lees, Pdes-mas: Distributed simulation of multi-agent systems, Proc. Comput. Sci. 18 (2013) 671–681.
[32] M. Scheutz, P. Schermerhorn, R. Connaughton, A. Dingler, Swages-an extendable distributed experimentation system for large-scale agent-based alife simulations, Proc. Artif. Life X (2006) 412–419.
[33] R.K. Standish, R. Leow, Ecolab: Agent based modeling for c++ programmers, 2004. ArXiv Preprint cs/0401026.
[34] L. Gasser, K. Kakugawa, Mace3j: fast flexible distributed simulation of large, large-grain multi-agent systems, in: Proceedings of the First Inter. Joint Conf. on Autonomous Agents and Multiagent Systems: Part 2, ACM, 2002, pp. 745–752.
[35] D.R. Hill, Parallel random numbers, simulation, and reproducible research, Comput. Sci. Eng. 17 (4) (2015) 66–71.
[36] B. Logan, G. Theodoropoulos, The distributed simulation of multiagent systems, Proc. IEEE 89 (2) (2001) 174–185. http://dx.doi.org/10.1109/5.910853.
[37] A.J. Cowling, H. Georgescu, M. Gheorghe, M. Holcombe, C. Vertan, Communicating stream x-machines systems are no more than x-machines, J. UCS 5 (9) (1999) 494–507.
[38] M. Holcombe, S. Coakley, R. Smallwood, A general framework for agent-based modelling of complex systems, in: Proceedings of the 2006 European Conference on Complex Systems, European Complex Systems Society, Paris, France, 2006.
[39] F. Cicirelli, L. Nigro, Control centric framework for model continuity in time-dependent multi-agent systems, Concurr. Comput.: Pract. Exper. (2016).
[40] M. Matsumoto, T. Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, ACM Trans. Model. Comput. Simul. (TOMACS) 8 (1) (1998) 3–30.
[41] G. Cordasco, R. Chiara, A. Mancuso, D. Mazzeo, V. Scarano, C. Spagnuolo, A framework for distributing agent-based simulations, in: Euro-Par 2011: Parallel Processing Workshops, Lecture Notes in Computer Science, vol. 7155, 2011, pp. 460–470.
[42] G. Marsaglia, A. Zaman, A new class of random number generators, Ann. Appl. Probab. (1991) 462–480.
[43] W. Hörmann, J. Leydold, G. Derflinger, Automatic Nonuniform Random Variate Generation, Springer, 2004.
[44] C. Márquez, E. César, J. Sorribes, A load balancing schema for agent-based spmd applications, in: International Conf. on Parallel and Distributed Processing Techniques and Applications, PDPTA, 2013.
[45] J. Himmelspach, A.M. Uhrmacher, Plug'n simulate, in: Proceedings of the 40th Annual Simulation Symposium, ANSS '07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 137–143. URL http://dx.doi.org/10.1109/ANSS.2007.34.
[46] U.I. Paulo Leitão, C.P. Rückemann, Parallelising multi-agent systems for high performance computing, in: INFOCOMP 2013: The Third International Conference on Advanced Communications and Computation, IARIA, Lisbon, Portugal, 2013.
[47] D. Pawlaszczyk, S. Strassburger, Scalability in distributed simulations of agent-based models, in: Proceedings of the 2009 Winter Simulation Conference, WSC, 2009, pp. 1189–1200.
[48] D. Weyns, H. Van Dyke Parunak, F. Michel, T. Holvoet, J. Ferber, Environments for Multiagent Systems State-of-the-Art and Research Challenges, Springer Berlin, Heidelberg, Berlin, Heidelberg, 2005, pp. 1–47.
[49] R. Červenka, I. Trenčanskỳ, M. Calisti, D. Greenwood, Aml: Agent modeling language toward industry-grade agent-based modeling, in: Agent-Oriented Software Engineering V, Springer, 2005, pp. 31–46.
[50] M. Frigo, S.G. Johnson, The design and implementation of fftw3, Proc. IEEE 93 (2) (2005) 216–231.
[51] G. Karypis, K. Schloegel, V. Kumar, Parmetis: Parallel Graph Partitioning and Sparse Matrix Ordering Library. Version 10, Dept of Computer Science, University of Minnesota, 1997.
[52] A.L. Chin, A.D. Worth, A.C. Greenough, A.S. Coakley, M. Holcombe, M. Kiran, Flame: An approach to the parallelisation of agent-based applications, Work 501 (2012) 63–259.
[53] X. Rubio-Campillo, Pandora: A versatile agent-based modelling platform for social simulation, in: SIMUL 2014, The Sixth International Conference on Advances in System Simulation, IARIA, Nice, France, 2014, pp. 29–34.
[54] X. Rubio-Campillo, J.M. Cela, Large-scale agent-based simulation in archaeology: an approach using high-performance computing, in: BAR International Series 2494. Proceedings of the 38th Annual Conference on Computer Applications and Quantitative Methods in Archaeology, Granada, Spain, April 2010, 2013, pp. 153–159.
[55] K.S. Perumalla, μsik-a micro-kernel for parallel/distributed simulation systems, in: 2005. PADS 2005. Workshop on Principles of Advanced and Distributed Simulation, IEEE, 2005, pp. 59–68.
[56] G. Cordasco, F. Milone, C. Spagnuolo, L. Vicidomini, Exploiting d-mason on parallel platforms: A novel communication strategy, in: Euro-Par Workshops (1)'14, 2014, pp. 407–417.
[57] P. Wittek, X. Rubio-Campillo, Scalable agent-based modelling with cloud hpc resources for social simulations, in: 2012 IEEE 4th International Conf. on Cloud Computing Technology and Science, (CloudCom), IEEE, 2012, pp. 355–362.
[58] B. Cosenza, G. Cordasco, R. De Chiara, V. Scarano, Distributed load balancing for parallel agent-based simulations, in: 2011 19th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, (PDP), IEEE, 2011, pp. 62–69.
[59] M. Madella, B. Rondelli, C. Lancelotti, A. Balbo, D. Zurro, X.R. Campillo, S. Stride, Introduction to simulating the past, J. Archaeol. Method Theory 21 (2) (2014) 251–257.
[60] S. Coakley, P. Richmond, M. Gheorghe, S. Chin, D. Worth, M. Holcombe, C. Greenough, Large-scale simulations with flame, Intell. Agents Data-Intensive Comput. 14 (123) (2015).

Please cite this article in press as: A. Rousset, et al., A survey on parallel and distributed multi-agent systems for high performance computing simulations, Computer Science Review (2016), http://dx.doi.org/10.1016/j.cosrev.2016.08.001