Simulation Modelling Practice and Theory 49 (2014) 320–335

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

New trends in parallel and distributed simulation: From many-cores to Cloud Computing ☆

Gabriele D'Angelo *, Moreno Marzolla

Department of Computer Science and Engineering, University of Bologna, Italy

ARTICLE INFO

Article history:
Received 7 February 2014
Received in revised form 17 June 2014
Accepted 18 June 2014
Available online 11 July 2014

Keywords:
Simulation
Parallel and distributed simulation
Cloud Computing
Adaptive systems
Middleware
Agent-based simulation

ABSTRACT

Recent advances in computing architectures and networking are bringing parallel computing systems to the masses so increasing the number of potential users of these kinds of systems. In particular, two important technological evolutions are happening at the ends of the computing spectrum: at the "small" scale, processors now include an increasing number of independent execution units (cores), at the point that a mere CPU can be considered a parallel shared-memory computer; at the "large" scale, the Cloud Computing paradigm allows applications to scale by offering resources from a large pool on a pay-as-you-go model. Multi-core processors and Clouds both require applications to be suitably modified to take advantage of the features they provide. Despite laying at the extreme of the computing architecture spectrum – multi-core processors being at the small scale, and Clouds being at the large scale – they share an important common trait: both are specific forms of parallel/distributed architectures. As such, they present to the developers well known problems of synchronization, communication, workload distribution, and so on. Is parallel and distributed simulation ready for these challenges? In this paper, we analyze the state of the art of parallel and distributed simulation techniques, and assess their applicability to multi-core architectures or Clouds. It turns out that most of the current approaches exhibit limitations in terms of usability and adaptivity which may hinder their application to these new computing architectures. We propose an adaptive simulation mechanism, based on the multi-agent system paradigm, to partially address some of those limitations. While it is unlikely that a single approach will work well on both settings above, we argue that the proposed adaptive mechanism has useful features which make it attractive both in a multi-core processor and in a Cloud system. These features include the ability to reduce communication costs by migrating simulation components, and the support for adding (or removing) nodes to the execution architecture at runtime. We will also show that, with the help of an additional support layer, parallel and distributed simulations can be executed on top of unreliable resources.

☆ An early version of this work appeared in [1]. This paper is an extensively revised and extended version in which significantly more than 30% is new material.

* Corresponding author. Address: Department of Computer Science and Engineering, University of Bologna, Mura Anteo Zamboni 7, I-40127 Bologna, Italy. Tel.: +39 051 2094511; fax: +39 051 2094510.

E-mail addresses: g.dangelo@unibo.it (G. D'Angelo), moreno.marzolla@unibo.it (M. Marzolla).

http://dx.doi.org/10.1016/j.simpat.2014.06.007
1569-190X/

Acronyms

ARTIS	Advanced RTI System
CMB	Chandy–Misra–Bryant
COTS	Commercial off-the-shelf
DES	Discrete Event Simulation
EOS	End of Step
GAIA+	Generic Adaptive Interaction Architecture
GPU	Graphics Processing Units
GVT	Global Virtual Time
HLA	High Level Architecture
HPC	High Performance Computing
HT	Hyper-threading
LCR	Local Communication Ratio
LP	Logical Process
MAS	Multi-agent system
MH	Mobile Host
PADS	Parallel and distributed simulation
PEU	Physical Execution Unit
SE	Simulated Entity
SIMD	Single Instruction Multiple Data
VSE	Virtual Simulated Entity
WCT	Wall Clock Time

1. Introduction

In the last decade, computing architectures were subject to a significant evolution. Improvements happened across the whole spectrum of architectures, from individual processors to geographically distributed systems.

In 2002, Intel introduced the Hyper-threading (HT) technology in its processors [2]. An HT-enabled CPU has a single execution unit but can store two architecture states at the same time. HT processors appear as two "logical" CPUs, so that the Operating System can schedule two independent execution threads. This allows a marginal increase of execution speed due to the more efficient use of the shared execution unit. Improvements of miniaturization and chip manufacturing technologies naturally led to the next step, where multiple execution units ("cores") are put on the same processor die. Multi-core CPUs can actually execute multiple instructions at the same time. These types of CPUs are now quite ubiquitous, being found on devices ranging from high-end servers to smartphones and tablets; CPUs with tens or hundreds of processors are already available [3]. The current generation of desktop processors often combines a multi-core design with HT. This means that each physical core will be seen by the Operating System as two virtual processors.

While many multi-core processors are homogeneous shared-memory architectures, meaning that all cores are identical, asymmetric or heterogeneous multi-core systems also exist. A notable example is the Cell Broadband Engine [4], that contains two PowerPC cores and additional vector units called Synergistic Processing Elements (SPE). Each SPE is a programmable vector co-processor with a separate instruction set and local memory for instructions and data. Other widely used asymmetric multi-core systems include General-Purpose Graphics Processing Units (GP-GPU) [5], which are massively parallel devices containing up to thousands of simple execution units connected to a shared memory. The memory is organized in a complex hierarchy, since RAM chips do not provide enough bandwidth to feed all cores at maximum speed. GP-GPUs are generally implemented as add-on cards which are inserted into a host computer. The host CPU then sends data and computation to the GPU through data transfers from host to device memory; results are fetched back to host memory when computation completes.

In modern multi-core processors, the level of complexity introduced by the hardware is quite high; unfortunately, this complexity cannot be ignored, and instead it has to be carefully exploited at the application level. In fact, writing efficient applications for multi/many-core processors requires a detailed knowledge of the processor internals to orchestrate communication and computation across the available cores.

Not only hardware is undergoing the major changes just described: at the software layer the Everything as a Service (EaaS) is gaining momentum thanks to Cloud Computing. According to NIST [6], Cloud Computing is a model for enabling on-demand network access to a shared pool of resources.

Depending on the type of resource which is provided, we can identify different Cloud service models. In a Software as a Service (SaaS) Cloud, customer receives access to application services running in the Cloud infrastructure; "Google Apps" is an example of a SaaS Cloud. A Platform as a Service (PaaS) Cloud provides programming languages, tools and a hosting environment for applications developed by the Cloud customer. Examples of PaaS solutions are AppEngine by Google,

Force.com from SalesForce, Microsoft's Azure and Amazon's Elastic Beanstalk. Finally, an Infrastructure as a Service (IaaS) Cloud provides its customers with low level computing capabilities such as processing, storage and networks where the customer can run arbitrary software, including operating systems and applications.

The architectural evolutions described above are not completely transparent to existing applications, which therefore should be modified to take full advantage of new features. In this paper, we focus on a specific class of software applications, namely simulation tools. Simulation can be very demanding in terms of computational resources, for example when dealing with large and detailed models [7]. Parallel and distributed simulation (PADS) [7] aims at studying methodologies and techniques for defining and executing simulation models on parallel and distributed computing architectures. PADS can be considered a specialized sub-field of parallel computing and simulation; as such, it shares many common issues with other types of parallel application domains (e.g., load balancing, partitioning, optimizing communication/computation ratio, and so on).

After a brief introduction on PADS, we review the most important challenges faced by PADS on multi-core processors and Clouds, and argue that it needs ad hoc solutions which are quite different from those commonly used to develop parallel or distributed applications. We see that, the most common PADS technologies are not adequate for coping with such architectural evolution. In particular, we analyze the limitations of current PADS approaches in terms of performance, functionality and usability. Moreover, we discuss the problem of finding new metrics for the evaluation of the PADS performance. In fact, in our view, such metrics have to consider many different aspects such as the execution speed, the cost of resources and the simulation reliability. Finally, we suppose that an adaptive approach can overcome many of these problems and therefore we describe a new parallel/distributed simulation technique that is based on the dynamic partitioning of the simulation model.

This paper is organized as follows. In Section 2 we give some background notions on parallel and distributed simulation. Section 3 deals with the challenges presented to PADS by many-core architectures and Clouds. In Section 4 we describe and discuss our proposal aimed to obtain more adaptable PADS. Finally, concluding remarks will be discussed in Section 5.

2. Background

A computer simulation is a software program that models the evolution of some real or abstract system over time. Simulation can be useful to evaluate systems before they are built, to analyze the impact of changes on existing systems without the need to physically apply the changes, or to explore different design alternatives.

In this paper, we focus on Discrete Event Simulation (DES) [8]: in a DES, the model evolution happens at discrete points in time by means of *simulation events*. Hence, the simulation model is evolved through the creation, delivery and execution of events. In its simplest form, a DES is implemented using a set of state variables, an event list, and a global clock representing the current simulation time [8].

A simulator capable of exploiting a single execution unit (e.g., a single CPU core) is called a *sequential* (monolithic) simulator. In a sequential simulation the execution unit processes events in non-decreasing timestamp order, and updates the model state variables and the global clock accordingly. The main advantage of this approach is its simplicity, but is clearly inappropriate for large and complex models. First, complex models might generate a large number of events, putting a significant workload on the CPU. Furthermore, the memory space required to hold all state information may exceed the RAM available on a single processor [9].

Parallel and distributed simulation (PADS) relies on partitioning the simulation model across multiple execution units. Each execution unit manages only a part of the model; in PADS each execution unit handles its local event list, but locally generated events may need to be delivered to remote execution units. Partitioning of the simulation model allows each processor to handle a portion of the state space and a fraction of the events; the parallel execution of concurrent events can also lead to a reduction of the simulation run time. Additional advantages of PADS include: the possibility to integrate simulators that are geographically distributed, the possibility to integrate a set of commercial off-the-shelf simulators, and to compose different simulation models in a single simulator [7].

A PADS is obtained through the interconnection of a set of model components, usually called Logical Processes (LPs). Therefore, each LP is responsible for the management of the evolution of a subset of the system and interacts with the other LPs for all the synchronization and data distribution operations [7]. In practice, each LP is usually executed by a processor core.

The scientific literature distinguishes between *parallel* and *distributed* simulation although the difference is quite elusive. Usually, the term *parallel simulation* refers to simulation modeling techniques using shared-memory multiprocessors or tightly coupled parallel machines. Conversely, *distributed simulation* refers to simulation techniques using loosely coupled (e.g., distributed memory) architectures [10]. It should be observed that, in practice, execution platforms are often hybrid architectures with a number of shared-memory multiprocessors connected through a LAN.

The lack of a global state and the presence of a network that interconnects the different parts of the simulator has some important consequences:

- The simulation model must be partitioned in components (the LPs) [11]. In some cases, the partitioning is guided by the structure and the semantics of the simulated system. For example, if the simulation involves groups of objects such that most interactions happen among objects of the same group, then it is natural to partition the model by assigning all the objects in the same group to the same LP. However, in the general case, it may be difficult if not impossible to identify an

easy way to partition the model. As a general rule, partitioning should guarantee that the workload is balanced across the available LPs, and the communication between the LPs is reduced [12–15]. Therefore, the partitioning criteria strictly depends both on the simulation model to be partitioned [16], on the underlying hardware architecture on which the model will be executed, and the characteristics of the synchronization algorithm that is implemented [17].

- The results of a parallel/distributed simulation must be identical to those of a sequential execution. This means that *causal consistency* [18,7] among simulation events must be guaranteed (see Section 2.1). While causal consistency can be trivially achieved in sequential simulations by simply executing events in non-decreasing timestamp order, it is much harder to achieve in PADS, especially on distributed-memory systems where LPs do not share a global view of the model. Therefore, PADS must resort to some kind of *synchronization* among the different parts that compose the simulator. Specific algorithms are needed for the synchronization of the LPs involved in the execution process.
- Each component of the simulator will produce state updates that are possibly relevant for other components. The distribution of such updates in the execution architecture is called *data distribution*. For obvious reasons, sending all state updates to all LPs is impractical. A better approach is to match the data production and consumption using some publish-subscribe method in which LPs can declare which type of data they are interested in and what they produce [19].

Partitioning, synchronization and data distribution are important problems in the context of PADS. Synchronization is of particular interest since parallel and distributed simulations tend to be communication bound rather than computation bound. Therefore, the choice of the synchronization algorithm, as well as the type of communication network used by the execution host, plays the most important role in determining the performance and scalability of the simulation model.

2.1. Synchronization

Implementing a parallel simulation requires that all simulation events are delivered following a message-based approach. Two events are said to be in *causal order* if one of them may depend on the other [18]. Enforcing causal order in a PADS requires that the different LPs are synchronized. That is because every LP can proceed at different speed, and therefore each partition of the simulation model may evolve at a different rate. Different solutions to this problem have been proposed, which can be broadly summarized in three main families: *time-stepped*, *optimistic* and *conservative* simulations.

In a *time-stepped synchronization*, the simulated time is divided in fixed-size intervals called *timesteps*. All LPs are required to complete the current timestep before moving to the next one [20]. The implementation of this approach requires a barrier synchronization between all LPs at the end of each step. Time-stepped synchronization is quite easy to implement, and it is most appropriate for models which already exhibit some form of lockstep synchronization (e.g., VLSI circuit simulation, where the clock signal is used to synchronize the functional blocks of the circuit). For other applications, it may be difficult to define the correct timestep size; furthermore, time-stepped mechanisms require that all newly generated events must be scheduled for future timesteps only (i.e., it is not possible to generate a new event to be executed during the current timestep), and this requirement can be too limiting for some applications.

The goal of the *conservative synchronization* approach is to prevent causality violations while allowing the global simulation time to advance at an arbitrary rate. Therefore, a LP can process an event with timestamp t only if no other event with timestamp $t' < t$ will be received in the future. The Chandy–Misra–Bryant (CMB) [21] algorithm can be used to guarantee causal delivery of simulation events in a distributed system. The CMB algorithm requires that each LP i has a separate incoming message queue Q_{ji} for each LP j it receives events from. Each LP is required to generate events in non-decreasing timestamp order, so that LP i can identify the next "safe" event to process by simply checking all incoming queues Q_{ji}: if all queues are nonempty, then the event with lowest timestamp t is safe and can be processed. If there are empty queues, this mechanism can lead to deadlock, which can be avoided by introducing null messages with no semantic content. The goal of these messages is to break the circular chain that is a necessary condition for deadlock. The main drawback of the CMB synchronization mechanism is the high overhead introduced by null messages in terms of both network load and computational overhead.

The *optimistic synchronization* approach does not impose the execution of safe messages only: each LP can process the events as soon as they are received. Causality violations can happen if some message (called a *straggler*) with timestamp in the past is received by some LP. When a causality violation is detected, the affected LP must roll back its local state to a simulation time prior to the straggler timestamp. The roll back mechanism must be propagated to all other LPs whose simulation time has advanced past that of the straggler [22,23]. This may trigger a rollback cascade that brings back the whole simulation to a previous state, from where it can be re-executed and process the straggler in the appropriate order. Optimistic synchronization mechanisms require that each LP maintains enough state data and a log of sent messages, in order to be able to perform rollbacks and propagate them to other affected LPs. A suitable snapshot mechanism must be executed periodically in order to compute a global state that is "safe", meaning that it cannot be rolled back. This is essential to reclaim memory used for events and state variables that become no longer necessary. This problem is known as the *Global Virtual Time* (GVT) computation, where the GVT denotes the earliest timestamp of an unprocessed message in an optimistic simulation. Computing the GVT is not unlikely taking a snapshot of a distributed computation, for which the well known Chandy–Lamport algorithm [24] can be used. However, in the context of PADS, more efficient ad hoc algorithms have been proposed [25].

2.2. Software tools

Many tools have been developed to support the implementation of PADS, some of which are described below.

μsik [26] is a multi-platform micro-kernel for the implementation of parallel and distributed simulations. The micro-kernel provides a rich set of advanced features such as the support for reverse computation and some kind of automated-load balancing. The last version of μsik was released in 2005 and now the development seems to have stopped. SPEEDES [27] and the WarpIV Kernel [28] have been used as testbeds for investigating new technologies, such as the Qheap, a new data structure for event list management. Furthermore, SPEEDES has been used for many seminal works on load-balancing in optimistic synchronization [29,30]. Finally, PRIME [31] and PrimoGENI [32] have specific focus on very high scalability and real-time constraints, mainly in complex networking environments.

One important advance in the field of PADS is the IEEE 1516–High Level Architecture (HLA) standard [33] that has been approved in 2000 and recently revised under the SISO HLA-Evolved Product Development Group [34]. HLA is a standard architecture for distributed simulation that allows computer simulations to interact with other simulations using standard interfaces.

The interest in IEEE 1516 has been initially very strong and it is still fundamental for interoperability and reusability of simulators. Despite of this, some drawbacks have been reported: the software architecture required by HLA [35], the lack of interoperability among different implementations [36], its complexity and steep learning curve [37] and, in some cases, poor execution time [38].

Table 1 shows some of the available (full or partial) implementations of the HLA specification. In general, as expected, commercial proprietary implementations are more compliant with the specification and feature-rich. However, the Portico Project implementation is very promising even if it still does not support the new HLA-evolved interface specification.

2.3. PADS on many-core and Cloud Computing platforms

The usage of many-core chips for running PADS is not a novel topic. Most of the existing works deal with the usage of optimistic Time Warp-based tools and their performance evaluation [50–52]. For example, in [50] it is shown that the architecture of many-core processors is rather complex and that some techniques are needed for increasing the execution speed (e.g., multi-level Time Warp, multi-level memory-aware algorithms, detailed frequency control of cores). The opportunity to control the speed of each CPU core is further investigated in [51]. In this case, the idea is that by controlling the execution speed of each core, it is possible to limit the number of roll-backs in optimistic synchronization. This approach is not different from adjusting the speed of CPUs, a widely investigated topic, but in this case the adjustment is done on different regions of the same chip. In [52] a multi-threaded implementation of an optimistic simulator is studied on a 48 core computing platform. This implementation, avoiding multiple message copying and reducing the communication latency, is able to reduce the synchronization overhead and improve performance. Finally, [53] shows how the Godson-T Architecture Simulator (GAS) has been ported to a many-core architecture with very good performance results but this has been possible mainly thanks to the loosely coupled nature of events in the specific simulation model of the GAS.

In [54] the authors evaluate the performance of a conservative simulator on a multi-core processor. Since the performance of conservative synchronization protocols can be affected by the large number of null messages generated to avoid deadlock conditions, the authors evaluate a number of known optimizations to reduce this overhead. A new hybrid protocol is finally proposed in order to combine the effectiveness of existing solutions.

A growing number of papers address the problem of running PADS on Cloud execution architectures. The early works on this topic have mainly considered the usage of Private Clouds [55–57]; more recent works deal with the usage of Public Clouds and the related problems [58–60]. In [61,62] the authors describe the state of the art of PADS on Cloud and propose a specific architecture based on a two-tier process partitioning for workload consolidation. In [63] an optimistic HLA-based simulation is considered and a mechanism for advancing the execution speed of federates at comparable speed is proposed. In [64], an approach based on concurrent simulation (that we have investigated in the past [65]) has been ported to the

Table 1
Some implementations of the HLA specification.

Name	Author	Bindings	Compliance	License
RTI NG Pro [39]	Raytheon	C++, Java	Full	Commercial
FDK [40]	Georgia Tech	C/C++	Partial	Non-free
MAK RTI [41]	VT MAK	C/C++, Java	Full	Commercial
Pitch pRTI [42]	Pitch technologies	Multiple	Full	Commercial
CERTI [43]	ONERA	Multiple	Partial	GPL/LGPL
OpenSkies RTI [44]	Cybernet systems	C++	Partial	Commercial
Chronos RTI [45]	Magnetar Games	C++/.NET	Unknown	Commercial
The portico project [46]	–	C++, Java	Partial	CDDL
SimWare RTI [47]	Nextel aerospace	C++	Partial	Commercial
Open HLA [48]	–	Java	Partial	Apache
OpenRTI [49]	FlightGear	C++, Python	Unknown	LGPL

Cloud. Finally, it is worth to mention [66] in which the authors propose to build simulations on the Cloud using handheld devices and a web-based simulation approach.

Even if many-core CPUs and Cloud Computing environments are very different execution architectures, with specific problems, it should be clear that the state of the art is, in both cases, made of quite specific solutions. Solutions that are tailored for a specific problem or a given simulation mechanism. In all cases with its own benefits and drawbacks.

3. Challenges

As already observed in the introduction, recent technological evolutions of computing systems require software developers to adapt their applications to new computing paradigms. Simulation developers, in particular those working on PADS, are no exception. In this section we discuss more specifically how multi-core processors and Cloud Computing affect simulation users.

The following issues that needs to be addressed by future PADS systems:

1. **Transparency**: parallel simulation systems should provide the possibility to hide low level details (e.g., synchronization and state-management issues) to the modeler, should the modeler be unwilling to deal manually with these issues.
2. **Simulation as a Service**: what are the main challenges that should be addressed in order to provide simulation "as a service" using a Cloud?
3. **Cost Models**: Cloud resources are usually provisioned on a pay-per-use pricing model, with "better" resources (e.g., more powerful virtual processors, more memory) having higher cost. Is this pricing model suitable for PADS?
4. **Performance**: Cloud Computing allows applications to dynamically shape the underlying computing infrastructure by requesting resources to be added/removed at run time. Cloud-enabled PADS systems should be extended to make use of this opportunity.

3.1. Transparency

Two of the main goals of the research work on PADS in the last decades were: (i) *make it fast*; (ii) *make it easy to use* [67]. Today, PADS can be very fast when executed in the right conditions [68], e.g., when the simulation model is properly partitioned, the appropriate synchronization algorithm is used and the execution architecture is fast, reliable and possibly homogeneous. In terms of usability, however, PADS does not yet work "out of the box".

In principle, the user of simulation tools should focus on modeling and analysis of the results. In practice, the choice of a specific tool (e.g., a sequential simulator of some kind) will preclude a future easy migration of the model towards another sequential simulator of a different kind, or towards a PADS system. For example, if a user wishes to migrate from a sequential to a parallel simulation, he will likely need to cope with such details as the choice of synchronization algorithm (optimistic or conservative), the allocation of simulated entities on Physical Execution Units, details of the messaging system that allows simulated entities to communicate, and so on.

Specifically, in PADS it is necessary to partition the simulation model across the available execution units. A good partitioning strategy requires that the communication among the partitions are minimized (cluster interactions) and the workload is evenly distributed across the execution units (load balancing). In case of static simulation models, i.e., models where the communication patterns among partitions do not change significantly over time, it is possible to statically define the partitions at model definition time. It should be observed that, even in the static scenario, identifying an optimal simulation model partitioning where inter-partition communication is minimized is a known NP-complete problem [69]. If the communication pattern changes over time, then some form of adaptive load balancing should be employed [70]. Adaptive load balancing is appealing since it can be applied also to static models to allow the "best" partitioning to be automatically identified without any specific application-level knowledge. Adaptive load balancing is difficult to implement (see [71] for an heuristic approach based on game theory), and is still subject of active research.

A better way to deal with the problems above would be to separate the simulation model from the underlying implementation mechanisms, which in the case of PADS means that low level details such as synchronization, data distribution, partitioning, load balancing and so on should be hidden to the user. This proved to be quite difficult to achieve in practice: an example is again the High Level Architecture (IEEE 1516) [33] that supports optimistic synchronization, but the low-level implementation of all the support mechanisms (such as rollbacks, see Section 2) is left to the simulation modeler [72]. Therefore, an existing conservative HLA simulation can hardly be ported to an optimistic simulation engine.

3.2. Simulation as a service

As already described in Section 2, some work on Cloud-based PADS has already been done [55–57]. In many of these works, Cloud technology is used for executing simulations in private Clouds. The goal is to enjoy the typical benefits of Cloud Computing (elasticity, scalability, workload consolidation) without giving up the guarantees offered by an execution environment over which the user has a high degree of control. However, the vast majority of simulation users do not

own a Private Cloud, therefore it would be much more interesting to run parallel simulations on a Public Cloud, using resources rented with a pay-per-usage model.

Obviously, the partitioning problem described above still holds; additionally, due to its nature, a Public Cloud provides a somewhat less predictable environment to applications. For example, execution nodes provided to users may be located in different data centers, have slightly different raw processing power, and be based on physical resources (e.g., processors) that are shared with other Cloud customers through virtualization techniques.

Considering the non-technical aspects, one of the potential advantages of the Public Cloud is the existence of multiple competing providers, that could – at least in principle – lead to a "market of services". In other words, the same service (e.g., computation) could be provided by many vendors and therefore the price is the result of a market economy of supply and demand. To take advantage of the competition among vendors, and the resulting price differences, the user must be able to compose services from different providers. The result is an heterogeneous architecture where performance and reliability aspects must be carefully taken into account.

However, the scenario above is unlikely, due to the lack of interoperability and the use of different APIs by different vendors. While in the short term vendor lock-in is beneficial to service providers, there already is pressure towards the use of open standards in Cloud Computing, especially in the academic/research community, culminated in the Open Cloud Computing Interface (OCCI) specifications [73]. It remains to be seen if and when open Cloud standards will be able to carve into the corporate community.

3.3. Cost models

The performance of a simulator is usually evaluated on the basis of the time needed to complete a simulation run (Wall-Clock-Time, WCT). However, this is not the only metric, and in some scenario it may also not be the most relevant one. For example, if the simulation is going to be executed on resources acquired from a Cloud provider, another important metric is the total cost of running the simulation, which is related both to the WCT and also to the amount of computation and storage resources acquired. Therefore, the user of simulation tools will have to decide:

- How much time he can wait for the results.
- How much he wants to pay for running the simulation.

The first constraint (how much time the user is willing to wait) should be always taken into consideration in any cost model. If no upper bound is set on the total execution time, then the cost of running the simulation essentially drops to zero, since that would allow the use of extremely cheap (if not totally free) resources of proportionally low performance. In practice, nobody is really willing to wait an arbitrary long time; hence a maximum waiting time is always defined, and this influences the cost of the resources that must be allocated to run the simulation: faster (and therefore expensive) resources are needed if the simulation result has to be provided quickly; slower (cheaper) resources can be employed if the user can tolerate a longer waiting time.

It is clear that, in a Public Cloud scenario, every computation and communication overhead should be minimized or at least carefully scrutinized. However, the two traditional synchronization mechanisms used in PADS (the Chandy–Misra–Bryant algorithm and the Time Warp protocol) have not been developed with these considerations in mind.

The Chandy–Misra–Bryant (CMB) algorithm [21] is one of the most well-known mechanisms used for conservative synchronization. Due to its nature, it requires the definition of some artificial events with the aim of making the simulation proceed. The number of such messages introduced by the synchronization algorithm can be very large [74,75]. Obviously, this communication overhead has a big effect on the WCT. Over the years, many variants have been proposed to reduce the number of such messages [76] and therefore reduce the communication cost. Despite of this, in some cases, the CMB synchronization can still be prohibitive under the performance viewpoint. On the other hand, the consideration that computation is much faster and cheaper than communication is at the basis of optimistic synchronization, e.g., the Time Warp protocol [22].

Both the conservative and optimistic synchronization mechanisms described above are not well suited for execution environments based on a "pay per use" pricing policy. Conservative synchronization is in general communication-bound and does not make effective use of CPUs. On the other hand, in a optimistic simulation, a very large part of the computation can be thrown away due to roll-backs. Very often the roll-backs do not remain clustered in a part of the simulator as they spread to the whole distributed execution architecture. This diffusion is usually implemented using specially crafted messages (called anti-messages) that consume a non-negligible amount of bandwidth. Finally, to implement the roll-back support mechanism it is necessary to over-provision many parts of the execution architecture (e.g., volatile memory). In a Public Cloud environment, all these aspects can have a very large impact on the final cost paid for running the simulation.

To summarize, the Public Cloud comes with a new cost model. Up to now most of the budget was for the hardware, the simulation software tools and writing the simulation model. Now it is no longer necessary to buy hardware for running the simulations, as computation is one of the many services that can be rented. If the goal is to have simulations that really follow the "everything as a service" paradigm, then we need some better way for building PADS, with mechanisms that need to be less "expensive", both in terms of computation and communication requirements. In this case, the main evaluation metric should not be the execution speed but the execution cost (or more likely a combination of both). A deeper look at the cost

model of Public Clouds (e.g. Amazon [77]) reveals some interesting facts. The pricing is often complex, with many different choices, configurations and options. For example, a new customer willing to implement a new service on top of the Amazon EC2 Cloud has to choose among several options: "On-Demand Instances[1]", "Reserved Instances" (Light, Medium or Heavy Utilization versions) and "Spot Instances". Furthermore, there are many instance types to choose from (e.g., General Purpose, Compute Optimized, GPU, Memory Optimized, Storage Optimized, Micro), each of them with many available options. Finally, for the dismay of many costumers, the price of each instance changes in the different regions (i.e., zones of the world). It is clear that each price and the detailed service specifications (e.g., the amount of free data transfer provided to each virtual instance) are part of a business strategy and therefore can change very quickly. Focusing more on the technical aspects, many of the "elastic" features (the possibility to dynamically scale up or down the resources (e.g. computation, memory, storage) provided by the Cloud have been built specifically for Web applications and therefore cannot be easily used for building simulators.

In [58], the authors analyze the cost optimization of parallel/distributed simulations run on the Amazon EC2 platform. More in detail, a cost to performance metric is defined and it used to find what EC2 instance type delivers the highest performance per dollar. In the paper, the most common EC2 instance types are analyzed under some assumptions (e.g. it is not considered that each partial instance-hour consumed is billed as a full hour). More in detail, the authors find that, under their metric, the best strategy is to use large (and costly) instances. This happens because in this kind of instance it is possible to cluster together many LPs and therefore minimize the network load.

3.4. Performance

We now focus our attention on the performance of a PADS implemented on a Public Cloud, when then goal is to obtain the results as fast as possible (i.e. the minimizing the WCT).

As usual, even if there are many aspects that could be considered, we focus on synchronization. Limiting the discussion on synchronization is not a concern given that if synchronization has poor performances then the rest of the simulator will not do much better. What would happen if the synchronization algorithms described in Section 2.1 are run on a Public Cloud without any modification? What performance can be expected?

Our analysis starts with the simplest synchronization algorithm: the time-stepped. As said before, the simulation time is divided in a sequence of steps and it is possible to proceed to the next timestep only when all the LPs have completed the current one. It is clear that the execution speed is bounded by the slowest component. This can be very dangerous in execution environments in which the performance variability is quite high. The whole simulation could have to stop due to a single LP that is slow in responding. This could happen for an imbalance in the model partitioning or for some network issues.

What about the Chandy–Misra–Bryant algorithm? We have already said that this algorithm is very demanding in terms of communication resources and that, in this case too, a slow LP could be the bottleneck of the whole simulation. The last possibility is to use an optimistic synchronization algorithm such as the Jefferson's timewarp [22]. This algorithm is not very promising either: timewarp is well-known to have very good performance when all LPs can proceed with an execution speed that is almost the same. This usually means that all LPs have to be very homogeneous in terms of hardware, network performance and load. Otherwise, the whole simulation would be slowed down by the roll-backs caused by the slow LPs. A requirement that is hard to satisfy in a Public Cloud environment. All these synchronization approaches have some characteristics that do not fit well with the Public Cloud architecture. A more detailed analysis shows that each approach has some pros and cons but the key problem is that all of them are not adaptable. It is just like working on dynamic problems with a static methodology.

In [58], the authors assess the performance and cost efficiency of different conservative time synchronization protocols (i.e. null-message sending strategies) when a distributed simulation is run on a range of Cloud resource types that are available on Amazon EC2 (i.e., different instance types). This work demonstrates that the simulation execution time can be significantly reduced using synchronization algorithms that are tailored for this specific execution environment and furthermore it shows that the performance variability, which is typical in low price instance types, has a noticeable impact on performance. Moreover, the authors foresee the adoption of dynamic forms of partitioning of the simulation model.

An interesting aspect of Cloud Computing is that it allows the application to shape the underlying execution infrastructure, rather than the other way around. This means that the application may require more resources (e.g., instantiate other computing nodes) or release them at run time. Focusing on computing power, a Cloud application may require a larger/lower number of computing nodes of the same type of those already running (*horizontal scaling*), or request to upgrade some or all the current computing nodes to a higher configuration, e.g., by asking for a faster processor or more memory (*vertical scaling*).

PADS could benefit from the above scaling opportunities, since they could be used to overcome the load balancing issues. Specifically, a simulator could request more computing nodes to reduce the granularity of the model partitioning (horizontal scaling), with the goal of reducing the CPU utilization of processor-bound simulations. On the other hand, for communication-bound models the simulator could consolidate highly interacting LPs on the same processor to replace remote communications with local ones. In this case, the simulator may request the Cloud to upgrade the nodes where the higher number of LPs are located (vertical scaling) in order to avoid the introduction of a CPU-bound bottleneck.

[1] In the Cloud Computing terminology, an instance is a virtual machine running some operating system and application software. A new instance can usually be obtained and booted in some minutes or less.

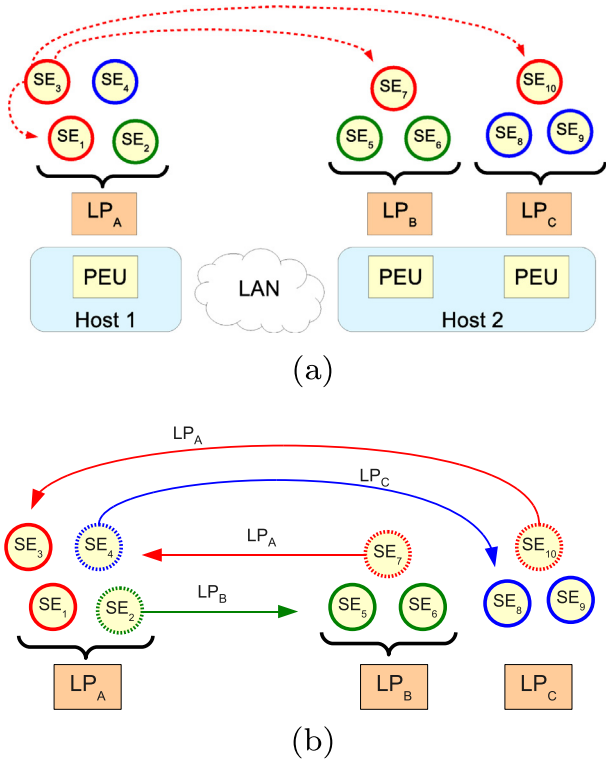


Fig. 1. A PADS with three LPs and ten SEs. (a) Initial situation and (b) after migration.

Deciding *when* and *how* to scale is still an open problem. For the *when* part, the user needs to define a decision procedure that tells when additional resources must be requested (*upscaling*) or when some of the resources being used can be relinquished to the Cloud provider (*downscaling*). Upscaling may be required to speed up the simulation, e.g., to meet some deadline on the completion time; upscaling may also be used during interactive simulations to focus on some interesting phenomena with a higher level of detail. Upscaling brings the issue of *how* to scale, i.e., choosing between horizontal and vertical scaling. The decision ultimately depends on the cost model being used (refer again to Section 3.3). If the user is only interested in maximizing the performance of PADS, then vertical scaling is more likely to provide benefits since it consolidates the workload on a lower number of more powerful nodes, thus reducing the impact of communication costs. If the cost model takes into consideration both performance and cost, then the choice between vertical and horizontal scaling becomes less obvious.

4. The quest for adaptivity

So far, we have analyzed some of the limitations of current PADS approaches. Before attempting to address these limitations, it is important to realize that there is no "silver bullet", i.e., no single solution that addresses them all in a comprehensive and coherent way. The last attempt to obtain a "one size fits all" approach to PADS resulted in the IEEE 1516 standard [33] that, as we mentioned, attracted several criticisms due to its complexity and limitations [35–38].

In this section we describe an approach to build scalable and adaptive simulation models by addressing the *partitioning problem* [15], that is the decomposition of the simulation model into a number of components and their allocation among the execution units. Our approach aims at achieving two goals: balance the computation load in the execution architecture and minimize the communication overhead [11]. If both these requirements are satisfied, then the simulation execution is likely to be carried out efficiently. The difficult part is that the balancing procedure should be transparent to users and adaptive. Adaptivity is of extreme importance given that both the behavior of the simulation model and the state of the underlying execution architecture cannot be accurately predicted.

In our view, the adaptive partitioning of the simulation model is a prerequisite for a solution to most of the PADS problems described in the previous sections.

4.1. Model decomposition

A complex simulation model should be partitioned in smaller parts referred to as Simulated Entities (SEs). Each SE interacts through message exchanges with other SEs to implement the expected behavior. We assume that the execution

environment consists of a set of interconnected Physical Execution Units (PEUs). For example, each PEU can be a core in a modern multi-core CPU, a processor in a shared memory multiprocessor, a node in a LAN-based cluster or even a Cloud instance. Following the PADS approach, the simulated model is partitioned among all the PEUs and each PEU is responsible for the execution of only a part of the model. In a traditional PADS, the model is partitioned in a set of Logical Processes (LPs) and each LP runs on a different PEU. Instead, in our case the LPs act as containers of SEs. In other words, the simulation model is partitioned in basic components (i.e., the SEs) that are allocated within the LPs. The SEs does not need to be statically allocated on a specific LP; indeed, each SE can migrate to improve the runtime efficiency of the simulator [78], as will be described shortly. For better scalability, new LPs can be allocated during the simulation, and idle ones can be disposed.

In practice, the simulation is organized as a multi-agent system (MAS) [79]. The MAS paradigm has the potential to enhance usability and transparency of simulation tools (as discussed in Section 3.1). Choosing the proper granularity of the SEs can be problematic. Having a large number of very simple entities will probably increase the communication and management overhead. Conversely, having only a few big entities means that the workload can be spread less effectively across the available PEUs, resulting in bad load balancing. The appropriate granularity of entities can sometimes be "suggested" by the simulation model itself; for example, it is pretty natural to model each node in a wireless network as a single SE.

4.2. Dynamic partitioning

To properly partition the model, we have to consider two main aspects. First, a PADS has to deal with a significant communication cost due to network latency and bandwidth limitations. Second, the execution speed of the simulator is bounded by its slowest component and therefore effective load balancing strategies must be implemented to quickly identify and remove performance "hot spots". As seen in Section 3.4, both the communication overhead and the computation bottlenecks have a big influence on the simulators performance (e.g., synchronization).

An effective strategy to reduce communication costs is to cluster the strongly interacting SEs together within the same LP [80]. However, a too aggressive clustering may result in poor load balancing if too many SEs are brought inside the same LP. Since communication patterns may change over time, the optimal partitioning is in general time-dependent, and is the result of a dynamic optimization problem with multiple conflicting goals and unknown, time-varying parameters.

Fig. 1a shows an example where a distributed simulation runs on two hosts connected by a local area network. Host 1 has a single PEU (e.g., a processor with a single core) and executes one LP, Host 2 has two PEUs executing one LP each. An LP is the container of a set of SEs that interact through message exchanges (depicted as dotted lines in the figure). The colors used to draw the SEs represent the different interaction groups, that is, groups of SEs that interact the most. The interaction groups are $\{SE_1, SE_3, SE_7, SE_{10}\}$, $\{SE_2, SE_5, SE_6\}$ and $\{SE_4, SE_8, SE_9\}$. If these interaction groups are expected to last for some amount of time, a more efficient allocation that reduces the communication overhead would be to migrate SEs as shown in Fig. 1b, so that each interaction group lies within the same LP. In this simple example, interaction groups are of approximately the same size, and therefore the new allocation is well balanced.

Since the interaction pattern between SEs may change unpredictably, it is necessary to rely on heuristics that monitor the communication and the load of each LP, and decide if and when a migration should happen. Obviously migrations have a non-negligible cost that includes the serialization of state variables of the SE to be migrated, the network transfer delay, and the de-serialization step required before normal operations can be restored at the destination LP.

The overall execution speed of the whole simulation is therefore the result of two competing forces: one that tries to aggregate SEs to reduce communication costs, and the other one that tries to migrate SEs away from overloaded LPs. Our experience with the practical implementation of adaptive migration strategies is reported in the following.

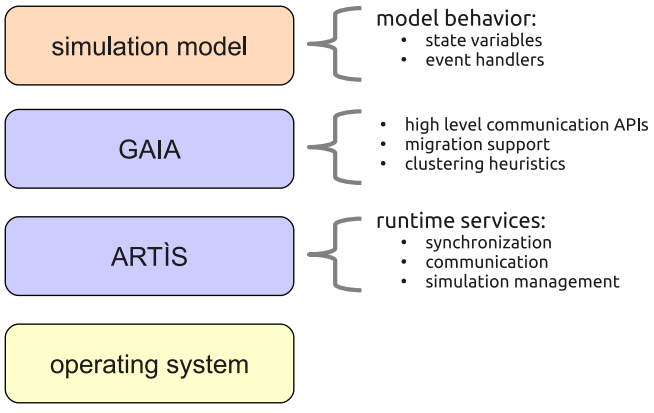


Fig. 2. High level structure of the simulator and its main components.

4.3. ARTÌS and GAIA+

In the last years, we pursued the approach described so far with the realization of a new simulation middleware called Advanced RTI System (ARTÌS) and the companion GAIA+ framework (Generic Adaptive Interaction Architecture) [15,78,80]. The high level structure of our simulator is shown in Fig. 2. The upper layer (labeled *Simulation Model*) is responsible for the allocation of the state variables to represent the evolution of the modeled system, the implementation of the model behavior and the necessary event handlers. The simulation model is built on top of the GAIA+ software framework, that provides the simulation developer with a set of services such as high level communication primitives between SEs, creation of SE instances and so on. GAIA+ can analyze the interactions (i.e., message exchanges) between SEs to decide if and how SEs should be clustered together, also providing the necessary support services. Given the distributed nature of a PADS, each LP analyzes the interactions of all SEs it hosts, and takes migration decisions based on local information and information received from other LPs, so that adaptivity can be achieved without resorting to any centralized component. Finally, in the layered architecture shown in the figure, the ARTÌS middleware [81] provides common PADS functionalities such as synchronization, low level communication between LPs, and simulation management services.

To validate ARTÌS and GAIA+, a number of models have been implemented, including wired and wireless communication environments [82,83]: using the ideas previously described, it has been possible to manage the fine grained simulation of complex communication protocols such as IEEE 802.11 in presence of a huge number of nodes (up to one million) [15]. In the wired case, we are working on the design and evaluation of gossip protocols in unstructured networks (e.g., scale-free,

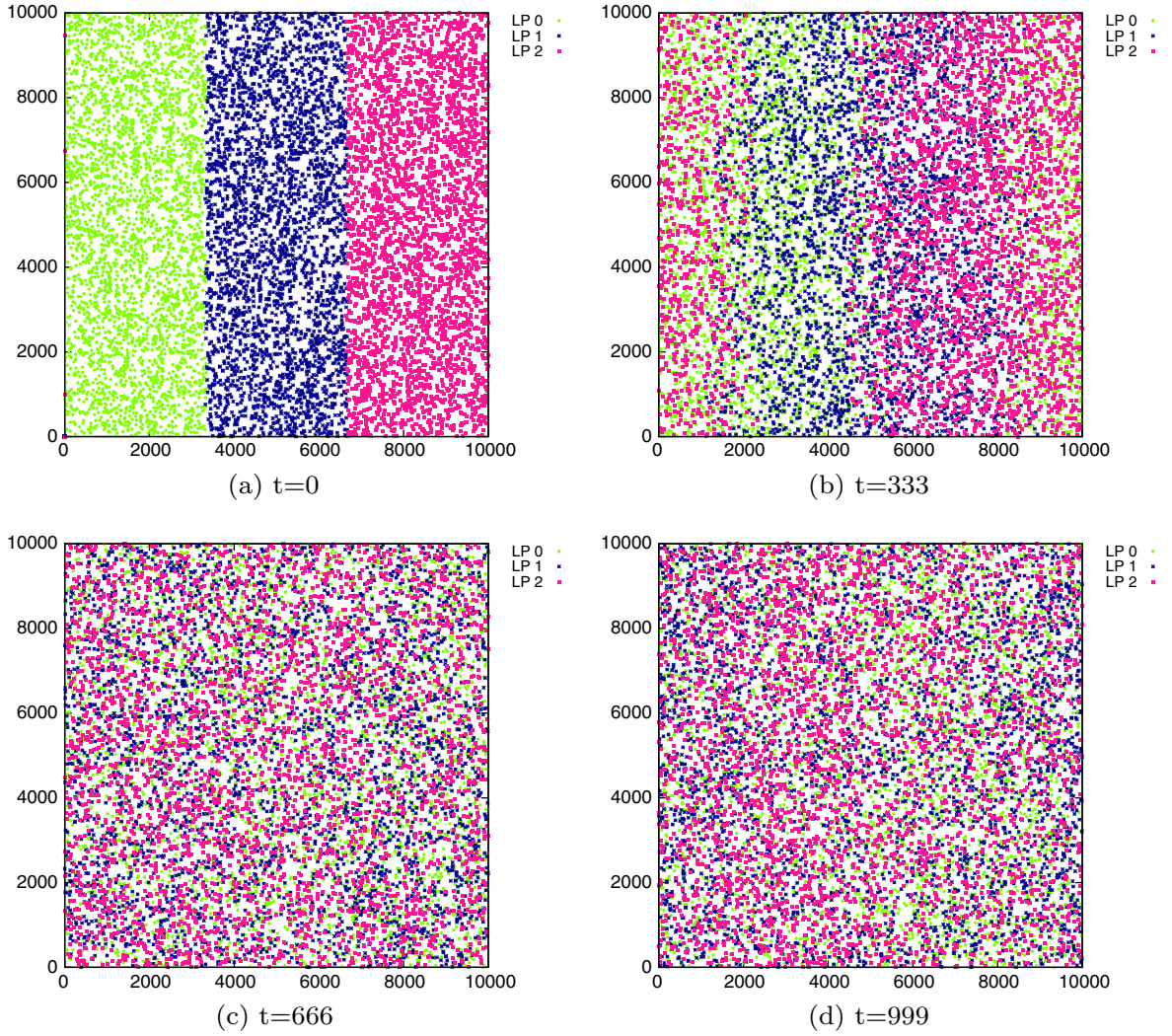


Fig. 3. Spatial position and allocation in LPs of the mobile wireless nodes simulation at different time steps. The color and shape of dots shows in which LP each node is allocated. Adaptive migration is not active (GAIA+ off). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

small-world, random) [84,85]. Good scalability has been achieved, using off-the-shelf hardware, thanks to the use of adaptive migration and load balancing techniques. Both this software tools have been designed to be platform-independent. In particular, the dynamic partitioning feature provided by GAIA+ fits very well with the horizontal and vertical scalability provided by Cloud Computing.

The ARTÌS middleware, the GAIA+ framework, sample simulation models, and the support scripts and scenario definition files are freely available for research purposes [81]. A large part of the software is provided in both binary and source form. We expect to make the source code for all components available through an open source license.

4.4. Case study

A complete evaluation of GAIA+/ARTÌS is outside the scope of this paper; however, a simple case study can be useful to better understand the proposed mechanism, and will be illustrated in this section. For the sake of simplicity, we consider only the adaptive clustering approach used to reduce the communication overhead. Advanced forms of load-balancing and reaction to background load have been implemented as well, but will not be examined here.

We consider a time-stepped, ad hoc wireless network model of 9999 Mobile Hosts (MHs). The simulated scenario is a two-dimensional area (10,000 × 10,000 units) with periodic boundary conditions, in which each MH follows a random way-point mobility model (*maxspeed* = 10 space–units/time–unit, 70% of nodes move at each timestep). The communication model is very simple and does not consider the details of low level medium access control. At each timestep, a random subset of 20% of the MHs broadcasts a ping message to all nodes that are within the transmission radius of 250 space units. This model captures both the dynamicity of the simulated systems and the space locality aspects of wireless communication. The MHs have been partitioned in 3 PEUs, each one running a single LP.

Our analysis starts with a scenario in which simulated entities do not migrate across LPs (the GAIA+ migration engine is turned off). To balance the workload evenly across the three LPs, each one should receive approximately 9999/3 = 3333 MHs. A simple metric that can be used to assess the ability to properly cluster the simulated components is the Local Communication Ratio (LCR), defined as the percentage of local messages with respect to the total messages sent or received by a simulated component (higher is better). A random assignment of MHs to LPs results in a LCR of $\frac{100}{\#LP}$ (%), where $\#LP$ as the number of LPs in the PADS. By taking into consideration the model semantics, we know that the communication between entities exhibits a strong spatial locality within the simulation area, since each MH only interacts with a local neighborhood. Therefore, a better assignment of MHs to LPs can be obtained by partitioning the simulated area, e.g., in vertical stripes, and to allocate all MHs in each stripe to the same LP, as shown in Fig. 3a. However, such allocation works well for the initial simulation steps, but then quickly degenerates as the spatial position of MHs change as the results of their movement. The situation can be observed in Fig. 3b–d, where the initial spatial locality of MHs is rapidly destroyed.

Fig. 4 shows what happens when the adaptive migration facility of GAIA+ is turned on, starting from a completely random allocation of MHs to LPs. GAIA+ analyzes the communication pattern of each MH and clusters the interacting hosts in the same LP. A snapshot at the end of the run (Fig. 4b) shows that the MHs have been clustered in groups, and that each group is determined by its position in the simulated area. It is interesting to observe that Fig. 4b is similar to what would be produced by Schelling's segregation model [86]; indeed, each MH has a preference towards the other MHs with which it communicates the most, and GAIA+ tries to cluster those MHs together within the same LP.

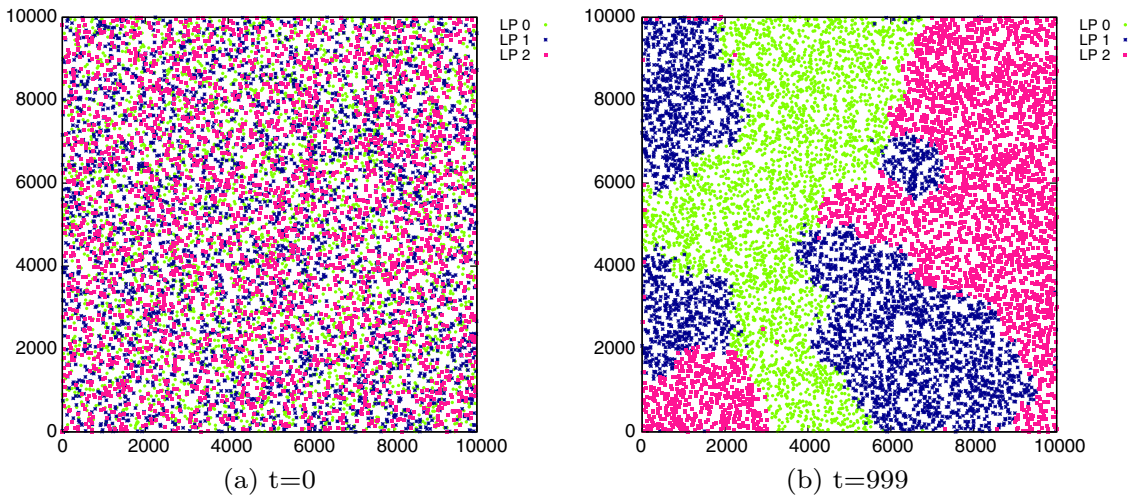


Fig. 4. Spatial position and allocation in LPs of the mobile wireless nodes at different time steps. The color and shape of dots shows in which LP each node is allocated. Adaptive migration is activated (GAIA+ ON). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The performance of the static allocation versus the dynamic allocation implemented by GAIA+ is shown in Fig. 5. The green line shows the mean LCR as a function of the simulated time step if MHs are statically assigned to the LPs at the beginning, while the red line shows the mean LCR when the adaptive migration of GAIA+ is enabled. Fig. 5a corresponds to the situation in which all MHs are initially randomly assigned to the LPs. As discussed above, the expected LCR in this situation is about 33%, which is the initial value at step $t = 0$. As the simulation proceeds, the LCR provided by the static allocation remains at the initial value. On the other hand, if the adaptive migration feature (GAIA+) is activated then the LCR rapidly increases.

In Fig. 5b we use the "sliced" allocation at $t = 0$, where MHs are assigned to LPs according to their initial position. In the case of static mapping (green line), the LCR drops to 33% as the movement of MHs destroys the initial statically imposed locality. If adaptive migration is used, the LCR remains stable at a nearly optimal value.

It is important to remark that a high LCR value does not guarantee, by itself, that a PADS scales well. In fact, migrating simulation entities does have a cost that can be significant if a large amount of state information needs to be moved from one PEU to another. However, previous studies have shown that good scalability can indeed be achieved even for models that do not exhibit the simple spatial locality of wireless ad hoc networks [87].

It is important to remark that the particular metric considered in this section (LCR) is independent from the execution environment (e.g., private clusters, Private or Public Clouds). In fact, a high LCR value demonstrates that the dynamic clustering of the simulated entities performed by GAIA+ leads to a reduction in the communication overhead since most communications are local. The actual impact of this reduction on the simulation wall-clock time depends on the (absolute) communication costs of the specific execution environment.

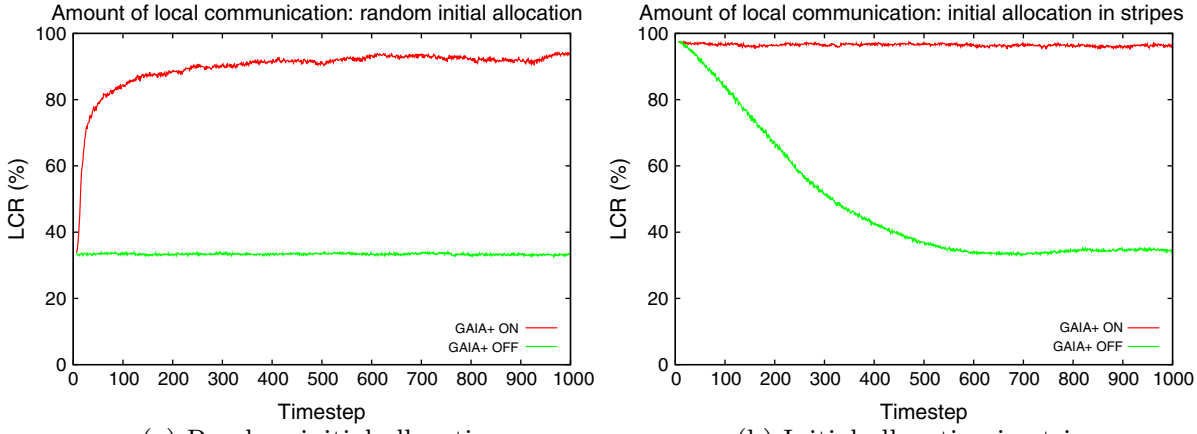
4.5. Fault tolerance

So far, we have described our effort in building adaptive PADS. In our opinion, this is a first step towards simulations that are able to run efficiently on modern computing infrastructures. A key point is still missing: the support for fault tolerance in distributed simulation. Usually, if a LP crashes (e.g., due to hardware failure of the underlying PEU), then the whole simulation aborts. In the case of a long-running simulation that is executed on hundreds or thousands of PEUs, the probability of a hardware failure during execution cannot be neglected. Therefore, fault tolerance is essential, especially if we wish to run large-scale simulations on cheap but unreliable execution systems such as resources provided by a Cloud infrastructure.

A new software layer called GAIA Fault-Tolerance (GAIA-FT) extends the interface provided by GAIA+ with provisions for fault tolerant execution. A GAIA-FT simulation is made by a set of interacting Virtual Simulated Entities (VSEs). A VSE is implemented as a set of identical SEs running on different LPs, therefore achieving fault tolerance through replication.

By selecting the number of replicas it is possible to decide how many crashes we want to tolerate, and also cope with byzantine failures. From the user's point of view, a VSE is just a special type of SE; in other words, the replication introduced by GAIA-FT is totally transparent and is built on the lower layers of the GAIA+ architecture. This means that the adaptive strategies described so far are still available and can be used to migrate replicas within a VSE to improve load balancing and communication. The basic implementation of GAIA-FT has been recently completed, and we are in the process of evaluating the replication mechanism.

Of course, fault tolerance comes at a cost, since replication increases the number of SEs to be partitioned. Therefore requiring more resources to execute the simulation, both in term of execution nodes and wall-clock time. This means that, a larger number of LPs will be used and that they will need to be kept synchronized in order for the simulation to advance, so the issues already described in Section 3.4 apply. Finally, when multiple copies of the same SE are created, the simulation



(a) Random initial allocation (b) Initial allocation in stripes

Fig. 5. Local Communication Ratio (LCR) evolution with different initial allocations.

middleware must ensure that all copies reside in LPs that are run on different execution nodes, to avoid that a failed processor brings down multiple replicas of the same SE. This makes the load balancing task more complex, since a new constraint is introduced (namely, multiple copies of the same SE must never reside on the same LP or on LPs that are run on the same execution node). Identifying a reasonable trade-off between reliability and performance in a parallel simulation is an interesting research topic that we are pursuing.

5. Conclusions

In this paper, we have reviewed the main ideas behind parallel and distributed simulation (PADS), and we observed that current PADS technologies are unable to fulfill many requirements (e.g. usability, adaptivity) in the context of the new parallel computing architectures.

Most computing devices are already equipped with multi-core CPUs, and Cloud-based technologies, where computation, storage and communication resources can be acquired "on demand" using a pay-as-you go model, are gaining traction very quickly. The "simulation as a service" paradigm has already been proposed as a possible application of Cloud technologies to enhance simulation tools.

We have shown that, the current PADS techniques are unable to fit well with these architectures, and that more work is needed to increase usability and performance of simulators in such conditions. We claim that a solution for such problems is required to deal with model partitioning across the execution nodes. To this aim, we propose an approach based on multi-agent systems. Its main characteristic is the adaptive migration of the simulated entities between the execution units. Heuristics can be used to reducing the communication cost and to achieve better load balancing of the simulation workload. Our proposal has been implemented in the ARTÌS/GAIA+ simulation middleware and tested with different models, with promising results.

Finally, the horizontal and vertical scaling possibilities provided by Cloud systems (see Section 3.4) deserve further investigations in the context of PADS. The Cloud Computing paradigm allows applications to request the type and amount of resources of their choice at run time, enabling dynamic sizing of the execution environment. How to enable a larger class of applications to take advantage of these possibilities is yes to be understood.

Acknowledgments

We would like to thank the anonymous reviewers whose detailed comments and suggestions greatly contributed to improve the overall quality of this paper.

References

[1] G. D'Angelo, Parallel and distributed simulation from many cores to the public cloud, in: HPCS '11: Proceedings of International Conference on High Performance Computing and Simulation, IEEE, 2011, doi:10.1109/HPCSim.2011.5999802.
[2] D.T. Marr, F. Binns, D.L. Hill, G.l. Hinton, D.A. Koufaty, A.J. Miller, M. Upton, Hyper-threading technology architecture and microarchitecture, Intel Technol. J. 6 (1), 2002.
[3] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, L. Bao, J. Brown, M. Mattina, C.-C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, J. Zook, TILE64 processor: a 64-core SoC with mesh interconnect, in: 2008 IEEE International Solid-State Circuits Conference, 2008, pp. 88–598, doi:10.1109/ISSCC.2008.4523070.
[4] M. Gschwind, Chip multiprocessing and the cell broadband engine, in: Proceedings of the 3rd Conference on Computing Frontiers, CF '06, ACM, New York, NY, USA, 2006, pp. 1–8. doi:10.1145/1128022.1128023.
[5] J.D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, J.C. Phillips, GPU computing, Proc. IEEE 96 (5) (2008) 879–899, http://dx.doi.org/10.1109/JPROC.2008.917757.
[6] P. Mell, T. Grance, The NIST Definition of Cloud Computing (Draft)–Recommendations of the National Institute of Standards and Technology, Special Publication 800-145 (draft), Gaithersburg (MD), Jan. 2011.
[7] R.M. Fujimoto, Parallel and Distribution Simulation Systems, first ed., John Wiley & Sons, Inc., New York, NY, USA, 1999.
[8] A.M. Law, D.M. Kelton, Simulation Modeling and Analysis, third ed., McGraw-Hill Higher Education, 1999.
[9] E. Egea-Lopez, J. Vales-Alonso, A. Martinez-Sala, P. Pavon-Mario, J. Garcia-Haro, Simulation scalability issues in wireless sensor networks, Commun. Mag., IEEE 44 (7) (2006) 64–73, http://dx.doi.org/10.1109/MCOM.2006.1668384.
[10] K. Perumalla, Tutorial: Parallel and Distributed Simulation: Traditional Techniques and Recent Advances, <http://kalper.net/kp/publications/docs/pads-fcrc07.pdf>, 2007.
[11] X. Zeng, R. Bagrodia, M. Gerla, GloMoSim: a library for parallel simulation of large-scale wireless networks, SIGSIM Simul. Dig. 28 (1) (1998) 154–161, http://dx.doi.org/10.1109/PADS.1998.685281.
[12] R.E. De Grande, A. Boukerche, Dynamic partitioning of distributed virtual simulations for reducing communication load, in: Haptic Audio visual Environments and Games, 2009, HAVE 2009, IEEE International Workshop on, 2009, pp. 176–181, doi:10.1109/HAVE.2009.5356113.
[13] H. Avril, C. Tropper, The dynamic load balancing of clustered time warp for logic simulation, in: Parallel and Distributed Simulation, 1996, Pads 96, Proceedings, Tenth Workshop on, 1996, pp. 20–27, doi:10.1145/238788.238804.
[14] P. Peschlow, T. Honecker, P. Martini, A flexible dynamic partitioning algorithm for optimistic distributed simulation, in: Principles of Advanced and Distributed Simulation, 2007, PADS '07, 21st International Workshop on, 2007, pp. 219–228, doi:10.1109/PADS.2007.6.
[15] G. D'Angelo, M. Bracuto, Distributed simulation of large-scale and detailed models, Int. J. Simulat. Process Modell. (IJSPM) 5 (2) (2009) 120–131, http://dx.doi.org/10.1504/IJSPM.2009.028625.
[16] E. Deelman, B. Szymanski, Dynamic load balancing in parallel discrete event simulation for spatially explicit problems, in: Parallel and Distributed Simulation, 1998, PADS 98, Proceedings, Twelfth Workshop on, 1998, pp. 46–53, doi:10.1109/PADS.1998.685269.
[17] A. Boukerche, S.K. Das, Dynamic load balancing strategies for conservative parallel simulations, in: Proceedings of the Eleventh Workshop on Parallel and Distributed Simulation, PADS '97, IEEE Computer Society, Washington, DC, USA, 1997, pp. 20–28. doi:10.1145/268826.268897.
[18] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Commun. ACM 21 (7) (1978) 558–565, http://dx.doi.org/10.1145/359545.359563.

[19] Y. Jun, C. Raczy, G. Tan, Evaluation of a sort-based matching algorithm for DDM, in: Proceedings of the Sixteenth Workshop on Parallel and Distributed Simulation, PADS '02, IEEE Computer Society, Washington, DC, USA, 2002, pp. 68–75. doi:10.1109/PADS.2002.1004202.
[20] S. Tay, G. Tan, K. Shenoy, Piggy-backed time-stepped simulation with 'super-stepping', in: Simulation Conference, 2003, Proceedings of the 2003 Winter, vol. 2, 2003, pp. 1077–1085, doi:10.1109/WSC.2003.1261535.
[21] J. Misra, Distributed discrete event simulation, ACM Comput. Surveys 18 (1) (1986) 39–65, http://dx.doi.org/10.1145/6462.6485.
[22] D. Jefferson, Virtual time, ACM Trans. Program. Lang. Syst. 7 (3) (1985) 404–425, http://dx.doi.org/10.1145/3916.3988.
[23] F. Quaglia, A. Santoro, Nonblocking checkpointing for optimistic parallel simulation: description and an implementation, IEEE Trans. Parallel Distributed Syst. 14 (6) (2003) 593–610, http://dx.doi.org/10.1109/TPDS.2003.1206506.
[24] K.M. Chandy, L. Lamport, Distributed snapshots: determining global states of distributed systems, ACM Trans. Comput. Syst. 3 (1) (1985) 63–75, http://dx.doi.org/10.1145/214451.214456.
[25] J.S. Steinman, C.A. Lee, L.F. Wilson, D.M. Nicol, Global virtual time and distributed synchronization, in: Proceedings of the Ninth Workshop on Parallel and Distributed Simulation, PADS '95, IEEE Computer Society, Washington, DC, USA, 1995, pp. 139–148. doi:10.1145/214282.214324.
[26] K.S. Perumalla, μsik – a micro-kernel for parallel/distributed simulation systems, in: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation, PADS '05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 59–68. doi:10.1109/PADS.2005.1.
[27] J.S. Steinman, J.W. Wong, The SPEEDES persistence framework and the standard simulation architecture, in: Proceedings of the Seventeenth Workshop on Parallel and Distributed Simulation, PADS '03, IEEE Computer Society, Washington, DC, USA, 2003, p. 11. doi:10.1109/PADS.2003.1207416.
[28] J.S. Steinman, C.N. Lammers, M.E. Valinski, K. Roth, K. Words, Simulating parallel overlapping universes in the fifth dimension with HyperWarpSpeed implemented in the WarpIV kernel, in: Proceedings of the Simulation Interoperability Workshop, SIW '08, 2008.
[29] L.F. Wilson, W. Shen, Experiments in load migration and dynamic load balancing in SPEEDES, in: Proceedings of the 30th Conference on Winter Simulation, WSC '98, IEEE Computer Society Press, Los Alamitos, CA, USA, 1998, pp. 483–490. doi:10.1109/WSC.1998.745025.
[30] L.F. Wilson, D.M. Nicol, Automated load balancing in SPEEDES, in: Proceedings of the 27th Conference on Winter Simulation, WSC '95, IEEE Computer Society, Washington, DC, USA, 1995, pp. 590–596. doi:10.1109/WSC.1995.478804.
[31] Parallel Real-time Immersive Network Modeling Environment – PRIME, <https://www.primessf.net>, 2014.
[32] N. Van Vorst, M. Erazo, J. Liu, PrimoGENI: integrating real-time network simulation and emulation in GENI, in: Principles of Advanced and Distributed Simulation (PADS), 2011 IEEE Workshop on, 2011, pp. 1–9, doi:10.1109/PADS.2011.5936747.
[33] IEEE 1516 Standard, Modeling and Simulation (M&S) High Level Architecture (HLA), 2000.
[34] 1516 WG – HLA Evolved Working Group, <http://standards.ieee.org/develop/wg/1516_WG.html>, 2014.
[35] W.J. Davis, G.L. Moeller, The high level architecture: is there a better way?, in: Proceedings of the 31st Conference on Winter Simulation: Simulation—A Bridge to the Future, WSC '99, vol 2, ACM, New York, NY, USA, 1999, pp. 1595–1601. doi:10.1109/WSC.1999.816898.
[36] J.-H. Kim, T. Kim, Proposal of high level architecture extension, in: T. Kim (Ed.), Artificial Intelligence and Simulation, Lecture Notes in Computer Science, vol. 3397, Springer, Berlin Heidelberg, 2005, pp. 128–137. doi:10.1007/978-3-540-30583-5_14.
[37] C. Boer, A. de Bruin, A. Verbraeck, Distributed simulation in industry – a survey part 3 – the HLA standard in industry, in: Simulation Conference, 2008, WSC 2008, Winter, 2008, pp. 1094–1102, doi:10.1109/WSC.2008.4736178.
[38] Z. Yuan, W. Cai, M. Low, S. Turner, Federate migration in HLA-based simulation, in: M. Bubak, G. Albada, P. Sloot, J. Dongarra (Eds.), Computational Science – ICCS 2004, Lecture Notes in Computer Science, vol. 3038, Springer, Berlin Heidelberg, 2004, pp. 856–864. doi:10.1007/978-3-540-24688-6_110.
[39] Raytheon RTI NG Pro, <http://www.raytheon.com/capabilities/products/rti/>, 2014.
[40] FDK – Federated Simulations Development Kit, <http://www.cc.gatech.edu/computing/pads/fdk/>, 2014.
[41] MAK Technologies, <http://www.mak.com/products/rti.php>, 2014.
[42] Pitch Technologies, <http://www.pitch.se/products/prti>, 2014.
[43] CERTI RTI, <http://savannah.nongnu.org/projects/certi>, 2014.
[44] OpenSkies – Cybernet, <http://www.openskies.net/features/features.html>, 2014.
[45] Magnetar Games – Chronos, <http://www.magnetargames.com/>, 2014.
[46] Portico Project, <http://www.porticoproject.org/>, 2014.
[47] SimWare RTI, <http://www.simware.es>, 2014.
[48] Open HLA, <http://sourceforge.net/projects/ohla/>, 2014.
[49] OpenRTI, <http://sourceforge.net/p/openrti/wiki/Home/>, 2014.
[50] K. Manian, P. Wilsey, Distributed simulation on a many-core processor, in: Proceedings of SIMUL 2011: Third Conference on Advances in System Simulation, IARIA, 2011.
[51] R. Child, P. Wilsey, Dynamically adjusting core frequencies to accelerate time warp simulations in many-core processors, in: Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation, PADS '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 35–43. doi:10.1109/PADS.2012.15.
[52] D. Jagtap, N. Abu-Ghazaleh, D. Ponomarev, Optimization of parallel discrete event simulator for multi-core systems, in: Parallel Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International, 2012, pp. 520–531, doi:10.1109/IPDPS.2012.55.
[53] H. Lv, Y. Cheng, L. Bai, M. Chen, D. Fan, N. Sun, P-GAS: parallelizing a cycle-accurate event-driven many-core processor simulator using parallel discrete event simulation, in: Proceedings of the 2010 IEEE Workshop on Principles of Advanced and Distributed Simulation, PADS '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 89–96. doi:10.1109/PADS.2010.5471655.
[54] S.D. Munck, K. Vanmechelen, J. Broeckhove, Revisiting conservative time synchronization protocols in parallel and distributed simulation, Concur. Comput.: Practice Exp. 26 (2) (2014) 468–490.
[55] R. Fujimoto, A. Malik, A. Park, Parallel and Distributed Simulation in the Cloud, SCS M&S Magazine (3), 2010.
[56] A. Malik, A. Park, R. Fujimoto, An Optimistic Parallel Simulation Protocol for Cloud Computing Environment, SCS M&S Magazine (4), 2010.
[57] S. Feng, Y. Di, Yuanchang, Z.X. Meng, Remodeling traditional RTI software to be with PaaS architecture, in: Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on, vol. 1, 2010, pp. 511–515, doi:10.1109/ICCSIT.2010.5564701.
[58] K. Vanmechelen, S.D. Munck, J. Broeckhove, Conservative distributed discrete-event simulation on the Amazon EC2 Cloud: an evaluation of time synchronization protocol performance and cost efficiency, Simulat. Modell. Practice Theory 34 (2013) 126–143. doi:http://dx.doi.org/10.1016/j.simpat.2013.02.002.
[59] S.B. Yoginath, K.S. Perumalla, Empirical evaluation of conservative and optimistic discrete event execution on cloud and VM platforms, in: Proceedings of the 2013 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, SIGSIM-PADS '13, ACM, New York, NY, USA, 2013, pp. 201–210. doi:10.1145/2486092.2486118.
[60] A. Aizstrauts, E. Ginters, D. Aizstrauta, P. Sonntagbauer, Easy communication environment on the cloud as distributed simulation infrastructure, in: Proceedings of the 5th WSEAS Congress on Applied Computing Conference, and Proceedings of the 1st International Conference on Biologically Inspired Computation, BICA'12, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 2012, pp. 173–178.
[61] X. Liu, Q. He, X. Qiu, B. Chen, K. Huang, Cloud-based computer simulation: towards planting existing simulation software into the cloud, Simulat. Modell. Practice Theory 26 (2012) 135–150, http://dx.doi.org/10.1016/j.simpat.2012.05.001.
[62] X. Liu, X. Qiu, B. Chen, K. Huang, Cloud-based simulation: the state-of-the-art computer simulation paradigm, in: Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation, PADS '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 71–74. doi:http://doi.ieeecomputersociety.org/10.1109/PADS.2012.11.

[63] Z. Li, X. Li, T.N.B. Duong, W. Cai, S.J. Turner, Accelerating optimistic HLA-based simulations in virtual execution environments, in: Proceedings of the 2013 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, SIGSIM-PADS '13, ACM, New York, NY, USA, 2013, pp. 211–220. doi:10.1145/2486092.2486119.
[64] M. Rak, A. Cuomo, U. Villano, mJADES: concurrent simulation in the cloud, in: Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on, 2012, pp. 853–860, doi:10.1109/CISIS.2012.134.
[65] L. Bononi, M. Bracuto, G. D'Angelo, L. Donatiello, Concurrent replication of parallel and distributed simulations, in: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation, PADS '05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 234–243. doi:10.1109/PADS.2005.6.
[66] E. Mancini, G. Wainer, K. Al-Zoubi, O. Dalle, Simulation in the cloud using handheld devices, in: Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012), CCGRID '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 867–872. doi:10.1109/CCGrid.2012.65.
[67] R. Fujimoto, Tutorial: Parallel & Distributed Simulation Systems: from Chandy/Misra to the High Level Architecture and Beyond, <http://22c118.cs.uiowa.edu/uploads/7/77/Fujimoto.pdf>, 2000.
[68] K.S. Perumalla, Scaling time warp-based discrete event execution to 104 processors on a Blue Gene supercomputer, in: Proceedings of the 4th International Conference on Computing Frontiers, CF '07, ACM, New York, NY, USA, 2007, pp. 69–76.
[69] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman & Co., New York, NY, USA, 1979.
[70] D. Nicol, R. Fujimoto, Parallel simulation today, Ann. Operat. Res. 53 (1) (1994) 249–285, http://dx.doi.org/10.1007/BF02136831.
[71] A. Kurve, C. Griffin, G. Kesidis, A graph partitioning game for distributed simulation of networks, in: Proceedings of the 2011 International Workshop on Modeling, Analysis, and Control of Complex Networks, Cnet '11, International Teletraffic Congress, 2011, pp. 9–16.
[72] A. Santoro, F. Quaglia, Transparent state management for optimistic synchronization in the high level architecture, Simulation 82 (2006) 5–20, http://dx.doi.org/10.1109/PADS.2005.34.
[73] Open Cloud Computing Interface (OCCI), <http://occi-wg.org/>, 2014.
[74] R. De Vries, Reducing null messages in Misra's distributed discrete event simulation method, IEEE Trans. Software Eng. 16 (1) (1990) 82–91, http://dx.doi.org/10.1109/32.44366.
[75] S.S. Rizvi, K.M. Elleithy, A. Riasat, Minimizing the null message exchange in conservative distributed simulation, in: Proceedings of International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering, 2006, pp. 443–448, doi:10.1007/978-1-4020-6266-7_80.
[76] W. Su, C.L. Seitz, Variants of the Chandy–Misra–Bryant Distributed Discrete-Event Simulation Algorithm, Tech. Rep., California Institute of Technology, Pasadena, CA, USA, 1988.
[77] Amazon EC2 Pricing, <http://aws.amazon.com/ec2/pricing/>, 2014.
[78] L. Bononi, M. Bracuto, G. D'Angelo, L. Donatiello, A new adaptive middleware for parallel and distributed simulation of dynamically interacting systems, in: Proceedings of the 8th IEEE International Symposium on Distributed Simulation and Real-Time Applications, IEEE Computer Society, Washington, DC, USA, 2004, pp. 178–187. doi:10.1109/DS-RT.2004.3.
[79] M. Wooldridge, An Introduction to Multiagent Systems, second ed., Wiley Publishing, 2009.
[80] L. Bononi, G. D'Angelo, L. Donatiello, HLA-based adaptive distributed simulation of wireless mobile systems, in: Proceedings of the Seventeenth Workshop on Parallel and Distributed Simulation, PADS '03, IEEE Computer Society, Washington, DC, USA, 2003, pp. 40–49. doi:10.1109/PADS.2003.1207419.
[81] Parallel And Distributed Simulation (PADS) Research Group, <http://pads.cs.unibo.it>, 2014.
[82] G. D'Angelo, S. Ferretti, Simulation of scale-free networks, in: Simutools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques, ICST, Brussels, Belgium, 2009, pp. 1–10, doi:10.4108/ICST.SIMUTOOLS2009.5672.
[83] L. Bononi, M.D. Felice, G. D'Angelo, M. Bracuto, L. Donatiello, MoVES: a framework for parallel and distributed simulation of wireless vehicular ad hoc networks, Comput. Networks 52 (1) (2008) 155–179, http://dx.doi.org/10.1016/j.comnet.2007.09.015.
[84] S. Ferretti, G. D'Angelo, Multiplayer online games over scale-free networks: a viable solution?, in: Proceedings of the 1th Workshop on Distributed Simulation and Online Gaming (DISIO 2010), 2010, pp. 5:1–5:8, doi:10.4108/ICST.SIMUTOOLS2010.8655.
[85] G. D'Angelo, S. Ferretti, M. Marzolla, Adaptive event dissemination for peer-to-peer multiplayer online games, in: Proceedings of the 2th Workshop on Distributed Simulation and Online Gaming (DISIO 2011), 2011.
[86] T.C. Schelling, Dynamic models of segregation, J. Math. Sociol. 1 (1971) 143–186.
[87] G. D'Angelo, S. Ferretti, LUNES: agent-based simulation of P2P systems, in: Proceedings of the International Workshop on Modeling and Simulation of Peer-to-Peer Architectures and Systems (MOSPAS 2011), IEEE, 2011. doi:10.1109/HPCSim.2011.5999879.