Adaptive parallel and distributed simulation of complex networks

Adaptive Parallel and Distributed Simulation of Complex Networks

Gabriele D'Angelo*

Department of Computer Science and Engineering
University of Bologna, Italy

Stefano Ferretti*

Department of Pure and Applied Sciences
University of Urbino "Carlo Bo", Italy

Abstract

Complex networks are an important methodology to model several (if not all) aspects of the real world, in which multiple entities interact, in some way. While many aspects related to such interactions can be investigated by looking at the general mathematical metrics of the networks, an alternative approach lies in the simulation of some application protocol on top of (large scale) complex networks. In this paper, we present a study on this intricate problem. The complexity of the simulation is due to the need to model all the interactions among network nodes. We focus on discrete-event simulation, a simulation methodology that enables both sequential (i.e. monolithic) and Parallel And Distributed Simulation (i.e. PADS) approaches. We discuss the performance and scalability requirements that the simulator should have. We also introduce a case study based on the agent-based simulation of gossip dissemination on top of a complex network. To demonstrate the viability of this simulation technique, we focus on a tool we built to simulate complex networks. The tool exploits adaptive partitioning mechanisms, which are essential to reduce the communication overhead in the PADS. An experimental

*Corresponding Author. Address: Department of Computer Science and Engineering. University of Bologna. Mura Anteo Zamboni 7. I-40127, Bologna. Italy. Phone +39 0547 338886, Fax +39 051 2094510

Email addresses: g.dangelo@unibo.it (Gabriele D'Angelo), stefano.ferretti@uniurb.it (Stefano Ferretti)

Postprint accepted by the Journal of Parallel and Distributed Computing June 3, 2024

evaluation has been conducted using different network topologies and simulator setups. Results demonstrate the feasibility of the approach to simulate complex networks.

Keywords: Simulation, Complex Networks, Parallel and Distributed Simulation

1. Introduction

Every aspect of life can be described as a set of interactions between an entity (being a human, any form of life, or even a synthetic system) and the external world, and other entities in particular. All these interactions among entities can be seen as creating a network of interactions. Most of these networks, built from the observation of interactions in the real world, exhibit complex connectivity patterns. To explore these patterns and study if and how they have an influence on the global system, such systems are described as graphs and analyzed using the mathematical theory of complex networks [40]. Each entity of the system is represented as a node while interactions among entities are mapped into links connecting the nodes of the graph. This process works no matter what the network is, e.g. a technological network (distributed system, the Internet of Things), a social network, a spreading disease, a biological network (metabolic and protein interaction networks), healthcare system [31]. The more complex the system, the more complex the resulting network [18].

The typical study of complex networks follows two possible approaches. The first one is to make a theoretical modeling of the system. In this case, a synthetic model is developed to analyze the system properties, by looking at some main metrics to be derived from the mathematical model itself. The second approach refers to a data science-based analysis. In this case, data traces are collected and manipulated to represent a real network as a graph. The study of the graph typically reveals some peculiarity of the network and general characteristics of its nodes. This is the common approach employed when dealing with social networks, for instance [37].

The problem arises when one would like to analyze the possible behavior of a system, knowing the behavior of its components and possible external factors that may alter the typical interactions within the system. Thus, for instance, it is of interest to understand whether changes in the system would alter its general properties. As an example, what would happen in a

social network, in presence of a novel filtering mechanism that hinders the dissemination of fake news? How an update on the road network would impact the traffic of a city? What happens if we deploy a set of sensors in a smart city and use the related information to control the flow of vehicles? What about trying to immunize a population during a pandemic, by planning ad-hoc vaccinations to most exposed humans?

As a matter of fact, one of the main problems is the lack of powerful tools for the simulation of complex networks. Under the simulation viewpoint, the modelling of large networks implementing complex interaction protocols is a challenging task. It is often difficult to build a simulator that is indeed able to mimic interaction protocols that are not naive and at the same time that can scale to a wide number of simulated entities. Thus, commonly used simulation tools are, on one hand, multi agent-based simulation systems which allow creating a high number of entities whose behavior is rather simple. Or, on the other hand, we do have small size simulators that only permit the creation of relatively small networks, i.e. a low number of simulated entities, in which nodes interact with moderate or low frequency. In essence, traditional simulation approaches permit building small-size toy models, or medium-scale models with simplified dynamics. In this paper, we discuss the main simulation techniques that cope with this issue.

The contribution of this paper is twofold. First, we show that through the use of simulation techniques it is possible to model complex dynamics over complex networks. This represents an alternative way of studying complex systems, with respect to analytical modeling or the study of static networks. The proposed framework works using Discrete Event Simulation (DES).

We also show how the use of Parallel And Distributed Simulation (PADS) strategies can truly improve the scalability of DES models. PADS allows modeling complex simulation environments, where a huge amount of discrete events is generated. In particular, the use of migration strategies allows moving simulation entities from one CPU core to another in the PADS system. This can strongly improve the performance of a parallel simulator. In fact, such schemes enable clustering highly interacting entities in the same CPU core, therefore reducing the communication overhead and increasing the scalability of the system.

Such an underlying framework is agnostic with respect to the simulation methodology employed to model the specific application or system under investigation. However, in this paper, we focus on multi-agent simulation systems. Agent-based simulation is largely employed in a variety of domains,

e.g. finance and economics, network modeling, epidemiology. In the case of complex networks, the simulation model can be specified by implementing the behavior of one or more categories of agents that represent the simulated entities. It is the interactions among agents that advance the simulation.

Second, we present LUNES (Large Unstructured NEtwork Simulator), a specific tool that implements the framework described above. LUNES is an easy-to-use tool for the generation and use of large graphs of whatever topology. It allows the expression of network dynamics through an agent-based simulation abstraction [44]. Moreover, it exploits parallelization and distribution features that allow building of scalable simulations. Through the description of LUNES, we show how agent-based simulation permits to easily model the interactions among network nodes and provides the granularity we need to properly perform load balancing of both computation and communication in parallel/distributed execution architectures. In fact, in our view, the agent is the smallest building block of the simulation model, i.e., the minimum set of variables that represent an entity. In this case, the concept of agent can be assimilated to the concepts of "object" or "entity" and even "node" in the case of network simulation.

To prove the viability of our first claim and assess the performance of the LUNES software, we conducted an experimental study specifically focused on scalability. In this assessment, we study different types of networks, such as random graphs and scale-free networks, with different simulator setups, e.g. a different number of CPU cores to be used by the simulator [40]. We measure how the simulator behaves, based on the number of simulated nodes and CPU cores in a parallel simulation. We also show that the use of adaptive partitioning improves the performance of the simulation, hence enabling large-scale simulations of complex networks. In particular, the application model we have used leads to a simulation that is strong communication bound, with a huge number of transmitted messages and a very low amount of computation on each node/entity. Thus, the most crucial aspect related to scalability are the communication protocol and the amount of messages, rather than the number of simulated nodes.

Obtained results demonstrate how the joint and combined use of agent-based simulation, Discrete Event Simulation (DES), Parallel And Distributed Simulation (PADS) strategies, equipped with adaptive migration approaches to balance the workload, allow to simulate application scenarios running on top of dynamic complex networks.

The remainder of this paper is organized as follows. Section 2 describes

the background about complex networks and simulation. Section 3 describes the LUNES simulator and Section 4 introduces the case study that has been used to obtain the results presented in Section 5. Finally, Section 6 provides some concluding remarks.

2. Background

In this section, we describe the main concepts at the basis of simulation and then we introduce discrete event simulation and parallel/distributed simulation techniques. We also briefly discuss agent based simulation. Then, we analyze the specific issues related to the simulation of complex networks.

2.1. Simulation

Computer simulation consists in the design of a software architecture that models the behavior of some system over time [22]. The simulated system can be a real system to be studied; alternatively, it might be something that has to be built, and its simulation helps in understanding which benefits and pitfalls the system being designed could have. Simulation can be used for several reasons, e.g. cost analysis, the need to test in a synthetic environment since doing such tests in a real environment would be too dangerous, or when many different solutions must be evaluated. From a technical viewpoint, the simulation of a system can be accomplished using a single process or using multiple networked components (e.g. CPU cores or hosts) that interact together to compute the whole simulated system.

In the last decades, some formalisms for modeling and analyzing general systems that can be discrete event systems have been proposed. Among them, the Discrete Event System Specification (DEVS) is quite popular in the field [50]. In this work, we are not interested in applying a particular formalism of description and in the following, we will describe the simulation background needed for the proposed methodology.

2.1.1. Discrete Event Simulation

Discrete Event Simulation (DES) [20] is an old but still very popular simulation paradigm due to its ease of use and good expressiveness level. A DES is represented by a simulated model, represented through a set of state variables, and its evolution, represented by a sequence of events that are processed in chronological order. Each event is timestamped, i.e. occurs

at a given instant in time, and represents a specific change in the state variables, i.e. that is a change of the simulated model state. Following this approach, the evolution of the whole simulated system is obtained through the processing of an ordered sequence of timestamped events. As an example, take intelligent transportation systems. In this case, the events are used to model the updates of the vehicles' positions, as well as data packet transmissions among vehicles. Thus, a DES can be implemented using a set of state variables (i.e. describing the modeled system), an event list (i.e. the pending events that will be processed for evolving the simulated state), and a global clock (i.e. the simulation time). It is important that each event is tagged by a timestamp that specifies the simulated time at which it has to be processed. Otherwise, the simulation would not model correctly the evolution of the simulated system. Furthermore, in order to embody some kind of randomness and variability to the model, the simulation must implement some pseudo-random choices. This way, through repeated simulation runs it is possible to study how the system behaves in different scenarios. This provides a comprehensive and clearer understanding of the simulated system and allows performing some statistical analysis on the obtained metrics.

2.1.2. Sequential DES

In a sequential, i.e. monolithic, simulation, a single Physical Execution Unit (PEU), which is a CPU core in case of a parallel simulation, is in charge of the generation of new events, managing and updating the pending event list that contains the future events and processing the events that are extracted in timestamp order from the list. Thus, the simulation is implemented as a single process that is executed on a specific CPU. The simplicity of the related software architecture has some drawbacks nevertheless. For instance, when the simulation consists of many different entities that are competing and interacting in a simulated world, having a sequential, single process program can make the task of simulating multiple and concurrent activities harder. Moreover, the main concern of this monolithic approach is about the simulator scalability, both in terms of execution time to complete the simulation runs and the complexity of the system that can be modelled [19].

2.1.3. Parallel DES and PADS

As an alternative to a sequential simulation, DES can be parallelized using a set of networked PEUs, e.g. CPU cores, processors or hosts. This approach

is usually referred to as Parallel Discrete Event Simulation (PDES) [11, 21, 22]. In this case, each PEU models only a part of the simulation model. This enables the modelling and the processing of larger and more complex simulation models with respect to DES. In fact, in massively populated simulation environments certain problems may become too big to be solved in a serial computing environment [7]. The memory and computational requirements cannot be supplied by a single CPU.

In PDES, each PEU manages a local pending events list. To let simulated entities interact with other simulated entities executed over other PEUs, some events are exchanged and delivered to these PEUs. As a consequence of this parallelization of the execution, a synchronization algorithm among PEUs is needed to guarantee the correct simulation execution [22].

Evidence demonstrates that, in many cases, a PDES approach can speedup the simulation execution [18]. In the rest of this paper, we will show that this is true also when simulating complex networks.

A possible extension to PDES is to employ more than a single multi-core CPU, thus offering the possibility to run the simulation over multiple CPUs, or even over multiple distributed computing nodes. This approach is commonly referred to as Parallel and Distributed Simulation (PADS) [22]. This approach can improve execution speed, model scalability, interoperability, and composability of different simulators. With respect to a sequential simulation, a PADS lacks a global model state, since each PEU manages a part of the simulated model only. According to the PADS terminology, the model components, executed on top of each PEU and interacting to advance the simulation, are called Logical Processes (LPs) [33].

More specifically, the implementation of a PADS requires that all simulation events are delivered following a message-based approach. The messages have the role to deliver the events exchanged between different LPs. Two events are defined to be in causal order if one of them may depend on the other [33]. As said above, the respect of causal order in a PDES is obtained by synchronizing all the LPs participating in a simulation. This is necessary since, for many different reasons, each LP can proceed at a different speed, and therefore each partition of the simulation model, represented in the specific LP, may evolve at a different speed [23]. Different solutions to the synchronization problem have been proposed. They can be broadly summarized as: time-stepped, optimistic [29], and conservative [11] synchronization.

Usually, the simulation is called "parallel" when the LPs are run on PEUs

with a shared memory that is available. Conversely, the simulation is "distributed" when the interconnection among the PEUs is given by a computer network, i.e. LAN, WAN, or Internet. Clearly enough, the performance of the underlying network has a significant effect on the simulator speed. Indeed, the parallelization/distribution of the simulator does not come for free. It is important to distribute the model as evenly as possible, doing whatever is possible to minimize message passing requirements between LPs and synchronizing all the LPs to ensure a correct and consistent simulation.

It is worth mentioning that distributed simulation is not only about performance, in fact in some cases interoperability is the most relevant subject. For example, in [48], the authors present a new transformation algorithm from workflow XML specification to G-DEVS models. In which, a workflow may involve a process model, different programs, and actors which are essential to its execution and the Generalised-DEVS (G-DEVS) is used to define abstractions of signals with piecewise polynomial trajectories. All that is obtained by means of the HLA specification that also allows the connecting of new HLA-compliant components in the Workflow environment. A seminal work for the interoperability in HLA is [49], in which is introduced DEVS/HLA, which is an HLA-compliant modeling and simulation environment that supports high-level model building using the DEVS (Discrete Event System Specification) methodology. The main goal of this approach is to provide a sound formal modelling and simulation framework.

2.1.4. Load-balancing in PADS and Adaptive PADS

In PADS, each LP handles a set of simulated entities [51]. This partitioning of the simulated model can be approached in many different ways. One possibility is trying to minimize the amount of network communication among LPs while doing the best for balancing the workload of the LPs in the parallel/distributed execution architecture.

Over the years, many static and dynamic approaches have been evaluated to automate and enhance the partitioning of parallel and distributed simulations. The most relevant partitioning approaches have been discussed in [17].

A typical approach is proposed in [51], where different partitioning schemes of the simulated region are used with the aim to assign an approximately equal number of simulated nodes to each partition and an equal number of partitions to each processor, therefore trying to reduce the amount of interprocessor messages. A relevant limitation of this approach is that, in this

case, the proposed partitioning schemes are static and cannot be adjusted at runtime and furthermore a static network topology is assumed.

Many of the approaches that have been proposed in the past to deal with load-balancing are designed to address computational load balancing [5] or the communication aspects [47, 14] but not both of them. In our view, this is a severe limitation of these approaches. Another relevant limitation is that, in many of the proposed approaches, the granularity of the load balancing mechanism is at the level of the whole LP. In other words, a whole LP is migrated from a CPU to another one. This is even more relevant in the simulation of complex networks in which representing each node with a different LP would lead to a huge number of LPs and therefore increasing the coordination and synchronization cost in the PADS.

More recently, in Distributed MASON [13], which is a scalable distributed multi-agent simulation environment, a novel geometric non-uniform work partitioning approach for the simulation environment has been proposed. In [9], the imbalance in computational load is considered and a generic self-adaptive framework is proposed. The framework is then evaluated when applied to two different simulation models: a micro-traffic simulation and a cellular automata simulation. In [36], the authors propose a new static resource allocation strategy based on the particle swarm optimisation algorithm for distributed simulations in cloud environments. Even if the proposed approach is static, it considers both computation and communication load. Finally, in [15] a novel bi-objective load balancing model is developed. This work is specifically aimed at the distributed High Level Architecture (HLA) simulation systems and it makes use of two meta-heuristic algorithms, i.e. NSGA-II and MOPSO, to avoid load imbalances in the distributed simulation.

In [17], we proposed a load-balancing approach in which the simulated model is represented by a multi-agent system. The simulated model is partitioned into small model components, also called Simulated Entities (SEs), and the model evolution is obtained through the exchange of interactions among SEs. In this way, the LPs are containers of SEs and it is possible to move (migrate) a SE from one LP to another. This permits avoiding the static partitioning of the simulated model and to adaptively reallocate the SEs for better computational and communication load balancing. In many cases, this leads to a speedup in the simulation execution and enhanced scalability. This adaptive PADS approach is implemented in the GAIA/ARTÌS simulator [3].

In the following of this paper, we will refer to two different versions of the adaptive PADS mechanism. The first one, which is referred as GAIA, implements a set of simple self-clustering heuristics that analyze the communication pattern of each SE and self-cluster them with the aim to minimize the communication cost [17]. Another, more complex mechanism, called GAIA+, implements the features of GAIA but also introduces a computational load-balancing scheme. More in detail, GAIA+ [17] checks the execution speed of each LP in the execution architecture and migrates some SEs with the aim to react to both imbalances in the simulated model (i.e. highly loaded nodes in the simulated network) and background load in the execution architecture (e.g. other processes running on the host/cluster used for running the simulation).

2.1.5. Agent-based Simulation

A particular type of simulation is the Agent Based Modeling and Simulation (ABMS) [42]. In ABMS, the simulated entities are called agents. Agents can represent any actor entity within a simulation [26]. The model specifies the behavior of each agent, i.e. microscale model; such behavior is usually influenced by the information that agents obtain from the environment [34].

Thus, events generated in the environment have an impact on the agent's states, and the whole simulation evolves based on these interactions among agents and the environment. Typically, the simulation model specifies one or more classes of agents; each agent of a certain class executes the same behavior procedure. The variety of the possible outcomes of a simulation is thus due to some kind of randomness introduced in the specification of such a behavior, and the different interactions and situations each agent is involved in.

2.2. Complex Networks and their Simulation

Complex networks are networks that feature patterns of connection between their elements. These patterns are usually neither purely regular, nor purely random. Many real-world networks, such as transportation systems, social, biological, communication networks, do have such complex patterns [4]. Complex network theory provides a useful methodology to understand the peculiarities of the whole network, identifying the most important entities, the critical elements, etc.

Whatever the nature of the dynamical processes under investigation, complex network theory provides a unified framework for their modeling [8, 27].

In essence, every entity being modeled is represented as a node of a network. Interactions among nodes are represented as edges of the network. We thus obtain a graph[1] that describes the dynamics of the specific process.

In this work, we are not specifically interested in such graphs management. There is a plethora of works that deal with many aspects of graph theory, in general [25, 32, 35]. However, these systems deal only with static graphs and do not consider the issue of processing evolving and dynamic graphs [6].

Rather, we are interested in the possibility of simulating and studying the complex dynamics of specific application models on top of such networks. Clearly, such dynamics might be strongly influenced by the topology of the underlying interaction network. To this aim, the most natural choice to model such systems is to resort to agent-based simulation, being each agent a node/entity of the network, and representing interactions between two entities as communications between agents. However, the agent-based approach poses several issues, such as hindering scalability, due to the number of agents to be simulated and their interactions. In particular, the amount of memory and computation to perform the simulation might result quite high and unevenly distributed in the execution architecture. In fact, using this modeling approach, each interaction between two nodes in the network results in a message exchanged between two simulated agents. Clearly enough, the simulation of an application protocol on top of a complex network might generate an important amount of messages to be handled. As a matter of fact, this is one of the main issues we consider in Section 5: Experimental Evaluation.

Take, for instance, a scale-free network, i.e., a network whose node degree[2] distribution follows a power law. This kind of network has hub nodes, i.e. nodes that have a degree much higher than others.

In general, a scale free network is characterized by a non-homogeneity of nodes, which is a key issue to be considered when designing simulators. This non-homogeneity in the number of connections causes different workloads at

[1]While the terms "graph" and "network" can be considered as synonyms in this paper, it is worth noting that often the word "graph" is used when referring to the mathematical object, i.e. a set of nodes with edges that connect them. Instead, the word "network" is used when a real system is modeled, through nodes that refer to some specific entity, and edges that represent interactions among nodes.

[2]The degree of a node is its amount of links in the network.

each node. These imbalances in the computation and communication load can lead to very unsatisfactory results, in the case of a naive partitioning of the network in a parallel or distributed execution architecture.

Several tools exist for the modeling of simple multi-agent systems that can be utilized to model a network of interactions. Examples are the multi agent simulation toolkit (MASON) [2], NetLogo [46], PeerSim [39], Swarm [38]. The rationale behind some of these tools, e.g. NetLogo and Swarm, is to provide an easy-to-use framework to develop toy models, where simplified agent behaviors can be rapidly implemented. The resulting analysis usually provides a good understanding of certain properties of the network, such as the presence of emergent behavior. However, typically such tools are not employed to model highly detailed simulation models, mainly for scalability issues.

PeerSim is a Java-based tool [39]. Its purpose is strictly confined to peer-to-peer systems, by modeling them as network overlays. Thus, the underlying idea is to look at these systems as a set of interacting nodes. Thus, the employed modeling methodology is agent-based. The tool has demonstrated good scalability and it has been widely employed to build simulators of peer-to-peer systems.

MASON is another Java-based tool, that offers easy-to-use APIs to build multi-agent simulations [2]. While the original tool is a centralized, single process tool, a new distributed version has been presented, i.e. DMASON [12]. With respect to these cited tools and the models they are typically able to support, our main goal is to simulate scenarios much more data-intensive, e.g. for the total number of messages that must be delivered in the network. In other words, we aim to deal with simulation models that cannot be addressed using a centralized approach. All cases in which the usage of multi-core CPUs, clusters of PCs or High-Performance-Computing (HPC) architectures are necessary.

Some works that focus on the use of PADS techniques to model and simulate complex networks are [16, 28, 41, 43]. In all the cited cases, the focus was solely on the simulation of scale-free networks. Usually, a node-based approach is utilized, i.e. each LP is in charge of the behavior of a subset of network nodes. Each interaction between two nodes causes a change state in the simulation. If the two nodes are within the scope of the same LP, that LP will handle the state change. Conversely, if the two nodes are managed by two different LPs, then a message exchange and coordination between the two LPs is required.

In contrast to the conventional approach of mapping each network node to an LP, in [41] a link partitioning scheme is employed, where each network link is mapped to a LP. The simulation handles information passing through each link, rather than focusing mainly on the behavior of the nodes. A flow of information is thus modeled and randomly simulated, traveling from sources to destinations. It is clear that this is a very different approach, not comparable with the idea of an agent-based simulation of complex networks, where nodes represent modeled entities.

The problem of computation and communication load-balancing in PADS has been extensively explored in the past. More specifically, we discussed the well-known problem of simulation model partitioning in our previous work [17]. To the best of our knowledge, LUNES is the most recent work that provides a series of advanced features for the simulation of complex networks.

3. The LUNES simulator

This section illustrates the aspects of the complex networks simulator that we designed and implemented. LUNES (Large Unstructured NEtwork Simulator) is an easy-to-use tool for the generation of large graphs of any topology [18]. It allows running a simulation model on top of the network nodes, with nodes that can interact through their interconnections, represented as network links. LUNES is based on a modular architecture, whose components are in charge of:

1. network creation;
2. implementation of the protocols;
3. analysis of the results.

The use of a modular approach permits the re-use and integration of existing software tools and facilitates the update and extensibility of the software. The flow of data processing goes as follows: a network overlay with a specific topology is created by the network generation module; then the protocol simulator executes the application/communication protocol on top of such a generated network; obtained results are processed by the trace analysis module. The application/communication protocol corresponds to the modeling of any kind of interaction that occurs between two or more entities during the simulation. If, for instance, the simulation model mimics a P2P overlay, the application/communication protocol mimics all the steps required to

exchange some data among nodes in the P2P system, i.e. the P2P communication protocol.

All the LUNES software architecture is designed and implemented for parallel/distributed processing. Therefore, all the LUNES modules can exploit the computational resources provided by parallel or distributed execution architectures. The interoperability between modules is obtained through simple template files, such as the graphviz dot language. In the following of this section, the LUNES modules are briefly described.

3.1. Network Topology Creation

LUNES is able to import the graph topologies generated by other tools. For example, in the current version, the graph topologies are generated by the *igraph* library [1]. This library provides algorithms for network analysis methods and graph theory and allows handling graphs with millions of vertices and edges. The graphs generated by igraph (or other tools) can be directly used for protocol simulation or stored in "corpuses", i.e. collections of homogeneous graphs. In this phase, the graph generated can be further annotated adding weights to specific edges or nodes. This would permit us to model and investigate setups and environments requiring that.

3.2. Protocol Simulation

LUNES provides to the simulation modeller a high-level Application Programming Interface (API) for the implementation of the protocols to be simulated. For example, in the current version all the most common features of dissemination protocols, used in peer-to-peer (P2P) distributed systems, are already implemented. Moreover, adding new variants or more complex protocols is straightforward. New protocols can be implemented using a set of available primitives. Thus, the simulationist can decide to avoid all the low-level simulation details. For example, LUNES has been used to easily develop an agent-based simulator of blockchains and then to study the effect of security attacks performed by malicious nodes.

To this aim, LUNES exploits the simulation services offered by the ARTÌS middleware and the GAIA/GAIA+ framework [3]. It is in these two tools that PADS features are implemented. Following the design of the simulation framework, LUNES represents each node in the simulated network as an agent. In the current implementation, there is no formalism behind the definition of agents and their interactions. In fact, they are implemented through very simple programs written using the C language.

More in detail, LUNES allows specifying the agent behaviour through the implementation of a behaviour function, as it is done in many multi-agent based systems. While the current LUNES implementation is based on the C programming language, any other language able to interoperate with C libraries, or even send messages through classic socket communication, can be exploited. As said before, each node in the simulated network is represented by an agent. The behavior of each agent is implemented through a set of function handlers that are able to both generate messages, to be delivered to other agents, and to change the agent's local state. Moreover, it is also possible to deal with dynamic topologies and to consider specific attributes (e.g. weights) to be added to the network edges. This permits the simulation of temporal networks.

3.3. Trace Analysis

Under the performance and scalability viewpoint, the most demanding points are the protocol simulation and the traces analysis. In LUNES, the trace analysis task has been designed as separated from the other simulation tasks; instead, some specific software tools have been implemented. The rationale behind this choice is that the simulation of a network, even with a few hundred nodes, can generate a huge amount of simulation traces that have to be stored, parsed, and analyzed. In LUNES, trace analysis is implemented using a set of shell scripts and some specific tools, for the computing of the required metrics on the trace data, that have been implemented in C language for efficiency. This mix is both quite efficient and easy to extend and personalize. We have intentionally avoided building a monolithic application to provide users with an easily customizable tool.

3.4. Time Evolution

LUNES exploits a time-stepped approach to perform simulations. This choice simplifies the deployment of the simulation over PADS architectures even if, in some cases, more complex conservative or optimistic synchronization algorithms can obtain better results in terms of execution speed. On the other hand, using a time-stepped synchronization allows exploiting the load balancing techniques and simulation entities migration approaches provided by the ARTÌS middleware and the GAIA/GAIA+ framework [3].

It is worth noticing that the use of time-stepped simulations is quite common when studying complex networks [10, 30] even if the simulation of

asynchronous systems using time-stepped simulations impose a neat granularity of the time steps. In fact, each timestep should be small enough to properly order and handle successive events that occur in time, thus guaranteeing a correct event ordering for subsequent events. Clearly, the choice of the timestep size has a relevant impact on the performance of the parallel/distributed simulator. In general terms, the larger the timestep size, the better is the efficiency of the synchronization mechanism. On the other hand, a timestep that is too coarse reduces the accuracy of the simulated model.

3.5. ARTÌS and GAIA/GAIA+

As described above, LUNES exploits the services provided by the ARTÌS simulation middleware and the GAIA/GAIA+ framework, which are briefly described in the following.

3.5.1. ARTÌS

The Advanced RTI System (ARTÌS) [3] is a parallel and distributed simulation middleware developed by our research group and partially inspired by the IEEE 1516 standard "High Level Architecture" (HLA). ARTÌS implements a typical parallel/distributed architecture in which the simulation model is statically partitioned in a set of LPs at bootstrap time. The goal of ARTÌS is to provide to the simulation model all the typical services of a distributed simulation middleware: communication, synchronization, and Data Distribution Management (DDM). The communication between the LPs is implemented efficiently by different communication protocols. The choice of the protocol depends on how the different PEUs are interconnected. For example, two LPs that are running on two CPUs cores in the same host communicate using low-latency and high-bandwidth shared memory, i.e. a specific implementation of shared memory to better suit the simulation requirements. Conversely, LPs running on hosts interconnected by the Internet communicate using TCP connections or reliable-UDP protocols like SCTP. In other words, the role of ARTÌS is to provide the best communication service in regards to the networking characteristics, i.e. latency, bandwidth, and jitter. For what concerns time management (i.e. synchronization), ARTÌS supports both conservative (Chandy-Misra-Bryant [11]) and optimistic [29]) synchronization algorithms. Moreover, a fully distributed implementation of the time-stepped synchronization is included. Finally, DDM is provided

by means of a very fast parallel implementation of the sort-based matching algorithm.

3.5.2. GAIA/GAIA+

The Generic Adaptive Interaction Architecture (GAIA) [3, 17] is a framework that exploits the services provided by ARTÌS. In GAIA, each LP is designed as the container of a set of simulated entities that can also be seen as agents. The simulated system behavior is then modeled by the interactions among the simulated entities. The enhancement provided by GAIA is based on the assumption that, in most cases, the interaction between the simulated entities is not uniform in terms of frequency and intensity. This means that it is often possible to find clusters of simulated entities in which "internal interactions" are more frequent than the "external ones", i.e. interactions with a destination that is outside the cluster. Obviously, in most simulated models (but not in all of them) the structure of these clusters changes over time. This effect is mainly caused by the simulation model evolution.

The basic task of GAIA is to identify such clusters with the aim to improve the performance of a PADS by allocating each cluster of heavily-interacting entities in the same LP. This operation, when successful, has the effect of reducing the amount of costly LAN/WAN communications with the more efficient shared memory messages. That clustering is not operated by GAIA at the simulation bootstrap since it would require the previous knowledge of the simulation model specificities and evolution but happens at runtime by means of entity migrations. In practice, GAIA analyzes the communication pattern of each simulated entity and implements a set of self-clustering heuristics to evaluate if the cost of migrating an entity is worth the communication enhancement provided by the clustering. We have developed GAIA with the aim to model a wide spectrum of systems. For this reason, all the provided heuristics are generic and not model-dependent. As an example, the basic clustering heuristics works as follows: every few simulation timesteps, for each simulated entity it is found which LP is the destination of the large percentage of interactions. If it is not the LP in which the simulated entity is contained and if the communication imbalance is relevant then a migration is triggered. The migration of simulated entities among LPs is totally transparent to the simulation model developer.

The main difference between GAIA and GAIA+ is in the form of load-balancing that is implemented. The aim of GAIA is to improve the communication efficiency in the parallel/distributed execution architecture while

preserving the current computational load in each LP. GAIA+ extends the previous heuristics by triggering adjunctive migrations with the aim to implement an adaptive computational load-balancing in parallel/distributed simulator. This is fundamental when in the presence of heterogeneous execution platforms in which the execution units are not identical, e.g. different CPUs and network performances. When necessary, GAIA+ can migrate some entities away from less powerful PEs towards more capable ones. Another case in which GAIA+ is useful is when the different simulated entities, in which the simulation model is partitioned, are not homogeneous. The adaptive computational load-balancing provided by GAIA+ is obtained by instrumenting the synchronization protocol implemented by the LPs. The LPs that are "slow" in terms of synchronization are then enabled by a distributed coordination protocol to trigger extra migrations, with respect to the migrations previously identified by GAIA, with the aim to balance the computation in the parallel/distributed execution by lifting the bottlenecks, i.e. the slow LPs.

4. Case study: gossip protocol dissemination

As a use case to show and test the peculiarities of LUNES in the simulation of a complex network, we implement a common communication protocol used in many distributed systems. In particular, we simulate an information dissemination protocol based on gossip running on a P2P overlay network [18]. This distributed system can be easily represented through multi-agent simulation. More specifically, the agents represent nodes of the P2P system, i.e. it is sufficient to implement the behavior of each agent (node) in the system, and then define the interactions that agents may have with other agents. In this case, each agent can interact with a subset of agents. This subset represents those nodes with which the considered node is connected with. Such connections define the P2P overlay, which can be a complex network of whatever topology. The global system behavior emerges from the single interactions among network nodes. In this use case, the behavior of the components of the system is rather simple: network nodes exchange application messages with their connected nodes, following the rules of a specific dissemination protocol.

It is worth pointing out that, in this simulated model, the P2P network is defined through the simple ability of agents to send messages to other agents. Thus, the overlay creation and management does not consider issues

concerned with the underlying physical network and proximity of nodes in the simulated distributed system. This is a common practice during the evaluation of a P2P protocol over an unstructured overlay [10, 30, 39]. Indeed, adding variables related to physical communication networks, such as network proximity and delay in message transmission, would increase the complexity of the model, hence making it more difficult to extract and compare some general results related to the performance of a dissemination protocol in an overlay.

All nodes generate a new message to be disseminated in the network. When the generation procedure is invoked at a given node, a single message is created with a certain probability, as described in Algorithm 1. The generation of a message simulates the occurrence of a new event produced at a given node that must be propagated. If the message is created, then it is sent through the net, using a DISSEMINATE() procedure (line 6 of the algorithm). The message is also inserted in a cache (line 5).

The caching scheme implements a classic Least Recently Used (LRU) approach that discards the least recently used messages first (code not shown in the algorithm).

Algorithm 1 Generation of a Message

```
1: function GENERATE()
2: t ← GENERATIONTHRESHOLD()
3: if RANDOM() < t then
4:     msg ← CREATEMESSAGE()
5:     CACHE(msg)
6:     DISSEMINATE(msg)
7: end if
```

Algorithm 2 Reception of a Message

```
1: function RECEIVE(msg)
2: if (NOTCACHED(msg) ∧ msg.ttl > 0) then
3:     CACHE(msg)
4:     msg.ttl ← msg.ttl − 1
5:     DISSEMINATE(msg)
6: end if
```

Algorithm 3 Dissemination: Gossip with Fixed Prob. of Dissemination (at n_i)

```
1: global γ                          {// global parameter shared among agents}
2:
3: function DISSEMINATE(msg)
4: for all n_j ∈ Π_i do
5:    if RANDOM() < γ then
6:       SEND(msg,n_j)
7:    end if
8: end for
```

Upon reception of a given message (see Algorithm 2), the receiving node forwards the message to its neighbors by calling the DISSEMINATE() function (line 5 in the algorithm). This is accomplished only if the message is not already in the node's cache. In fact, if the message is in the cache, it has already been disseminated; hence, the node has nothing to do with the message *msg* (line 2). Conversely, *msg* is transmitted and cached (line 3 of Algorithm 2). Needless to say, due to the possible memory constraints of a node, the cache is limited in size.

The DISSEMINATE() function is described in Algorithm 3. It is implemented as a *fixed probability* scheme. Let denote with n_i the node executing the function. This node randomly selects nodes to which the message *msg* has to be propagated [24, 45]. In particular, all n_i's neighbors (i.e. Π_i) are considered and a threshold value $\gamma \leq 1$ is maintained, which determines the probability that *msg* is gossiped to the neighbor. Hence, on average, at each step, the message is propagated from n_i to $\gamma|\Pi_i|$ other nodes. Clearly enough, the higher the node degree, the higher its workload.

4.1. Simulation Model Characterization

As already mentioned, the simulation model implemented in LUNES that will be used in this performance evaluation is based on a data dissemination protocol. Two different network topologies will be investigated: random networks and scale-free topologies. Both of them are very common in complex networks. As described previously, the network generation process in LUNES is obtained by means of the igraph network library. Table 1 reports the specific library functions used to generate the network graphs on which the fixed-probability dissemination protocol is executed. As we will

see in the remainder of this section, this communication protocol produces a huge amount of messages that need to be delivered and processed. It is worth noticing that many of the delivered messages are duplicates of original messages. To cope with this issue, with the aim to improve the efficiency of the gossip protocol, two adjunctive mechanisms are usually implemented: message cashing and Time-To-Live (TTL). More in detail, each node implements a cache that stores the identifiers of the messages following the policy described above. Furthermore, all the generated messages in the dissemination are set with a TTL that limits the number of hops that each message can traverse.

igraph generators		
	method	nodes
Random	igraph_erdos_renyi_game	100-3000
Scale-free	igraph_barabasi_game	100-3000

Table 1: igraph generators used for building the networks.

Under the simulation modeling viewpoint, the main characteristics to be considered are: a) the number of nodes in the simulated network, b) the number of edges, c) the number of delivered messages and, finally, d) the computational load of each node in the network. The number of simulated nodes has an effect on both the total number of original messages (i.e. that are not duplicated messages due to forwards), that are generated in the network, and on the computational load that is caused by updating the node variables and implementing the local cache. The effect caused by the number of edges is on the amount of duplicated messages that are generated during the dissemination. In fact, when a given node receives a new message i.e. that is not already in the local cache then the fixed-probability dissemination protocol evaluates for each edge of the node if the message must be forwarded or discarded. As said by the name of this gossip approach, this evaluation is randomized and based on a fixed-probability threshold. It is clear that increasing the number of edges in the network leads to an overall increase of the duplicated messages generated by the forwarding nodes. The effect of the number of messages, both original and duplicated, on the simulator performance is two-fold and it is the most important parameter to be considered in the scalability evaluation of the simulator. On one side, each message

needs to be processed by the simulator and delivered. In a PADS setup, both the latency and bandwidth constraints can strongly reduce the performance of the simulator. On the other side, each incoming message triggers in the receiving node the processing of the message (e.g. the execution of the fixed-probability gossip protocol) and more specifically a lookup operation in the data structures used by the message caching mechanism. Under the computational viewpoint, these operations can be quite costly. All this can hugely increase the simulation execution time.

As mentioned, in this simulated model the main part of the computational load is due to the generation of new messages and the forwarding of received ones. For this reason, in this performance evaluation, we model the dissemination protocol without adding any synthetic load to the nodes, while many real-world applications such as, for instance, the simulation of blockchain networks, would require a larger amount of data to be processed by nodes. The effect of our decision is that in this performance evaluation the parallel setups are disadvantaged with respect to the sequential approach, and distributed ones would be even more. The reason for this behavior is very simple: the PADS approaches are able to parallelize the model computation at the cost of an increased communication cost while in the setup described above there is a huge amount of communication and a limited amount of computation. In other words, this simulated model is communication bound and therefore we can not expect very large performance gains by employing a parallel setup and we should expect relevant slowdowns in presence of distributed execution architectures.

Table 2 reports the main model and simulator parameters for setting up the experiments reported in the following. To foster the reproducibility of our experiments, all the source code used in this performance evaluation and the raw data obtained in the experiments are freely available on the research group website [3].

Figure 1 reports the total number of messages that are delivered during a single simulation run when considering an increasing number of nodes and links per node. As discussed above, we consider this metric as the main aspect for the scalability of the simulator to be considered. All the results reported in this performance evaluation are obtained averaging the data collected in multiple independent runs. This specific result shows that the network topology (random vs. scale-free) has a negligible impact on the number of generated messages. On the other hand, the network size (i.e. number of nodes) has a big impact on the number of messages. In other words, the num-

Model and Simulator parameter	Description / Value
Number of nodes	[100,3000]
Number of delivered messages	see Figure 1
Simulated time	1000 timesteps
Time-To-Live (TTL)	= network diameter
Dissemination protocol (gossip)	fixed probability
Dissemination probability (gossip)	0.8
Prob. of each node to generate a new message	0.01 (per timestep)
Prob. of nodes that are generators of messages	1 (i.e. all nodes)
Prob. of nodes that are forwarders of messages	1 (i.e. all nodes)
Cache size (positions) in each node	256

Table 2: Simulation model and Simulator parameters.

ber of messages delivered in the network increases exponentially with respect to the number of nodes. That exponential increase in the number of delivered messages makes it impossible, or very slow, to simulate networks composed of a very large number of nodes while using this specific configuration of the simulation model. On the other hand, it would be sufficient to reduce the number of messages generated by each node to allow the simulation of large-scale setups but this is not the aim of this paper.

Another preliminary aspect that needs to be investigated is what is the effect of the number of links per node on the total amount of delivered messages. Figure 1 shows the results obtained when considering the scale-free network topology when adding, during the preferential attachment, 2, 3, and 4 links per node. As described above, due to the characteristics of the dissemination protocol, increasing the number of links in the network has the effect of boosting the number of duplicate messages that are delivered during each simulation run. When not stated differently, the considered networks in the following of this performance evaluation are those obtained by adding 2 links, during the preferential attachment, for each node.

This result shows how the simulation of a message dissemination strategy represents a complex use case for PADS, due to the simplicity of the computation performed at LPs, and the high message generation rate. This means that the scalability of the PADS execution, in this specific use case, is ruled by the ability to optimize local interactions within LPs, and minimize the

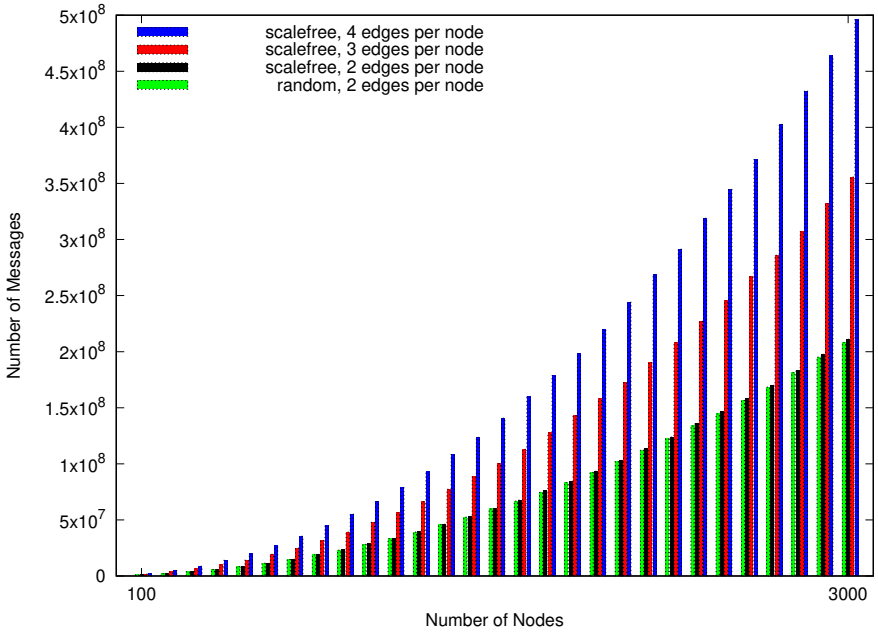

Figure 1: Number of delivered messages in a single simulation run with an increasing number of Nodes and Links per node. Random and Scale-free network topologies.

inter-LP communications. In other words, an efficient parallel/distributed execution is possible only by means of strategies that are able to reduce/limit the communication overhead necessary for a correct PADS implementation.

5. Experimental evaluation

The main goal of this section is to evaluate the scalability of LUNES in presence of different setups of the execution platform used for running the simulator and the parameters used to instantiate the complex network simulation model. More specifically, we are interested in evaluating the scalability of the simulator with respect to an increasing and very large number of delivered messages (see Figure 1).

In this performance evaluation, the main metric that is measured and discussed is the amount of time that is necessary for the simulator to complete the simulation of the dissemination protocol on the network overlay. In other words, other phases such as the network topology generation and the trace analysis (previously described in Section 3) are not considered since they are not necessary for addressing our research question. The hardware

CPU	Intel Xeon E5-2640
Clock frequency	2.00 GHz
Processors	2
Cores/proc	8
Total cores	16
Threads/core	2
HyperThreading	Yes
RAM	128 GB
L3 cache size	20480 KB
Operating System	Ubuntu 18.04 LTS
Kernel	4.15.0-66-generic (x86_64)

Table 3: Execution architecture used for the experimental evaluation.

platform and the software configuration used in this performance evaluation are reported in Table 3.

Given that the simulated model used in this performance evaluation is communication-bound, the following performance evaluation has been run only on a parallel machine. Running the gossip-based dissemination protocol on top of a distributed architecture, i.e. a LAN or Internet-based cluster can be very interesting. However, under the performance viewpoint, the latency introduced by the distributed communication architecture would be so relevant to nullify the effect of the computational parallelization given by the set of computation units. In other words, the distributed simulator would be slower with respect to both the parallel and the monolithic setup.

5.0.1. Random graph topology

We start the performance evaluation by measuring the amount of time (wall-clock time, WCT) that is necessary to complete a sequential run of LUNES (i.e. $LPs = 1$) in presence of an increasing number of simulated nodes organized following a random graph topology. This means that, in this case, the simulator is run on a single CPU without exploiting any parallelization feature that is provided by the multiple execution cores that are available.

As expected, Figure 2 shows that the sequential setup (i.e. the black line) is unable to deal with networks composed of a large number of nodes. As expected, the scalability of a monolithic simulator when running a complex network model is quite limited. The sharp increase in the WCT follows

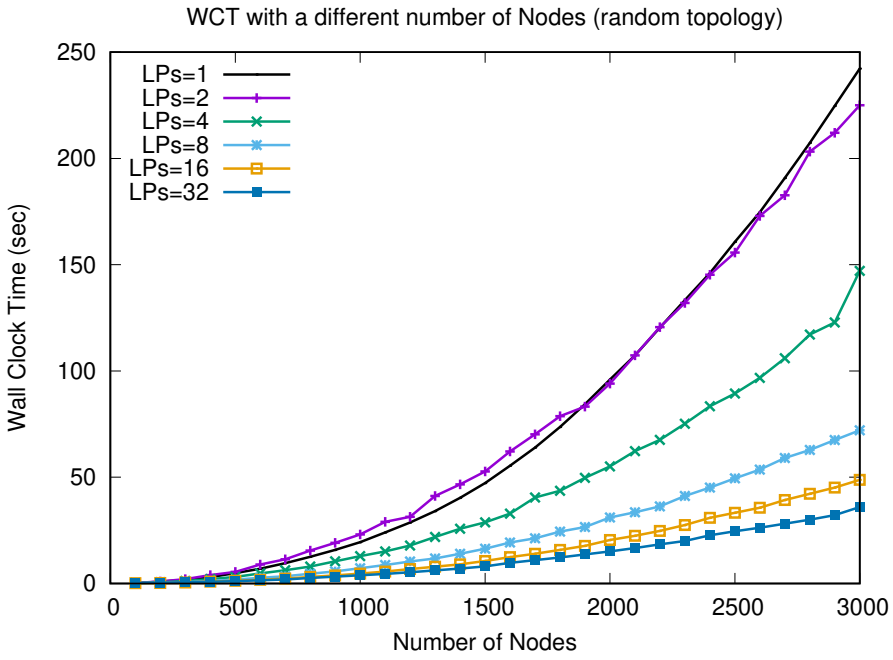

Figure 2: Wall-Clock-Time of a simulation run (random network topology) in sequential and parallel setups (i.e. GAIA OFF) with an increasing number of Nodes. Lower is better.

the behavior of the dissemination protocol in terms of messages that are delivered in the system (i.e. Figure 1). The more the messages are generated and duplicated in the overlay network, the more the simulator is unable to cope with the load. The result is more and more time to complete each simulation run. In Figure 3 is reported the efficiency in terms of speedup, that is the ratio between the sequential WCT and the WCT of a specific parallel setup, of the different simulation configurations with an increasing number of nodes. From the figure, it is clear that a configuration with 2 LPs is unable to provide a speedup that can be achieved by increasing the amount of computational resources. Overall, the speedup that can be obtained with 16 and 32 LPs is relevant but it is mainly due to the inefficiency of the sequential run when in presence of a large number of nodes (see Figure 2).

In a parallel simulation (i.e. $LPs \geq 2$), the simulated nodes are partitioned among the different LPs. In this paper, we consider the typical case in which the simulated nodes are evenly distributed in the execution architecture. This means that at the simulation bootstrap the simulated entities

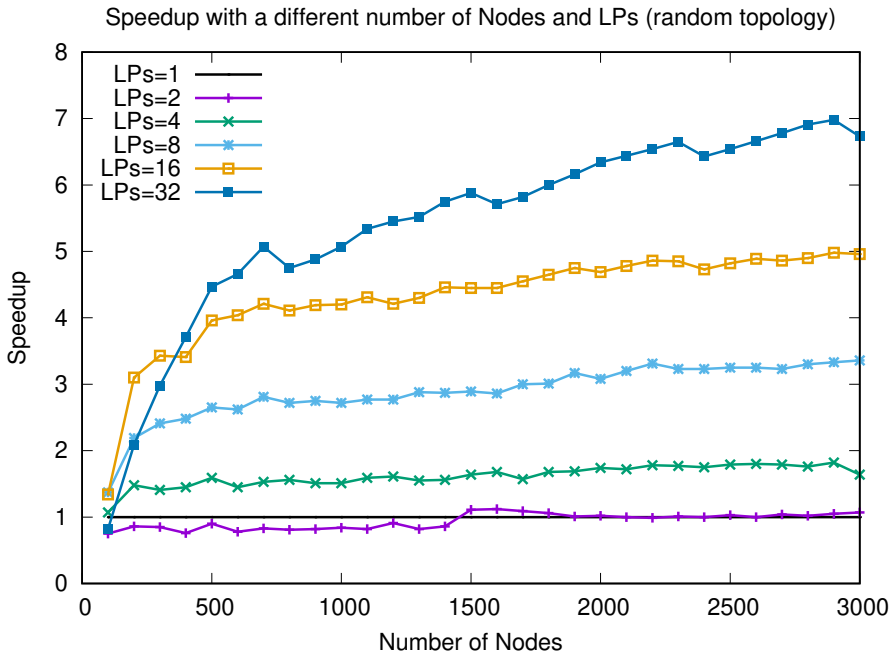

Figure 3: Speedups in sequential and parallel setups (i.e. GAIA OFF) with an increasing number of Nodes (random network topology). Higher is better.

are equally split among the LPs. This allocation is static, i.e. it will last for the whole simulation run. Conversely, when adaptive partitioning techniques are used (i.e. GAIA/GAIA+ is enabled), the number of simulated entities in each LP will dynamically change during the simulation when reallocations are triggered by GAIA/GAIA+ for load-balancing purposes.

We start investigating what happens when using 2 LPs (i.e. $LPs = 2$). The simulator is able to parallelize the computation at the cost of an increased communication cost. This is shown by the purple line (i.e. $LP = 2$): when the number of nodes is less than 1900 the parallel setup is slower than the sequential one. This is due to the fact that the parallelization provided by using 2 PEUs is not sufficient to compensate for the extra cost caused by the communication between the two execution units used by the simulator. On the other hand, for a larger number of nodes, there is a negligible gain. The gain of the parallel setup is significant only when the number of LPs is increased. In fact, in this case, the communication cost is more than balanced by the computation load sharing given by the multiple PEUs that are used.

This trend is confirmed when using 16 LPs, in which the gain is very relevant. Further increasing the number of LPs to match the virtual cores (i.e. $LPs = 32$) provided by the Hyper-threading technology, that is available in the processors equipping the testbed, slightly increases the simulator speed with respect to $LPs = 16$. This happens because a Hyper-threading CPU core has a single execution unit but can store two architecture states at the same time. In fact, Hyper-threading CPUs cores are detected by the operating system as two "logical" CPU cores. In practice, the operating system and then the simulator can schedule two independent execution threads. This allows a marginal increase of execution speed that is due to increased efficiency in the management of the shared execution unit but is not comparable with the execution speed obtained by "physical" CPUs cores.

The results show that the best performance of a PADS, in terms of speed of simulation, depends on many different factors. Not always the larger the execution architecture the better the performance. There is a trade-off between the parallelization/distribution of the workload among different LPs and the number of messages that need to be transmitted, in order to synchronize all the LPs. In other words, it is a case-by-case evaluation and the performance of the simulator is very difficult to predict before running it.

Figures 4 and 5 report the effect of the GAIA on the WCT in different setups, i.e. an increasing number of LPs used to parallelize the computation. In this case, the adaptive mechanism has the goal to reduce the communication overhead among the LPs while the computational load balancing mechanism provided by GAIA+ is disabled. In all LPs configurations that have been measured, GAIA is able to improve the simulator execution efficiency by clustering communications. The effect is a reduction of the WCT. The data shown in Figure 5 demonstrate that the best configuration for this simulation model, in terms of WCT, is obtained with 32 LPs and when GAIA is enabled. In the considered setup with 32 LPs, the self-clustering adaptive mechanism (i.e. GAIA) is able to provide a gain but this is very thin if compared to the setup with 8 or 16 LPs. The reason behind this behavior is related to the partitioning of the simulation model that is operated by the PADS approach. In fact, partitioning the model entities into more and more parts increases the complexity of the self-clustering operations and reduces its efficiency. In particular, increasing the number of LPs increases the probability that a group of frequently interacting entities needs to be allocated to different LPs. For example, to fulfill the load-balancing con-

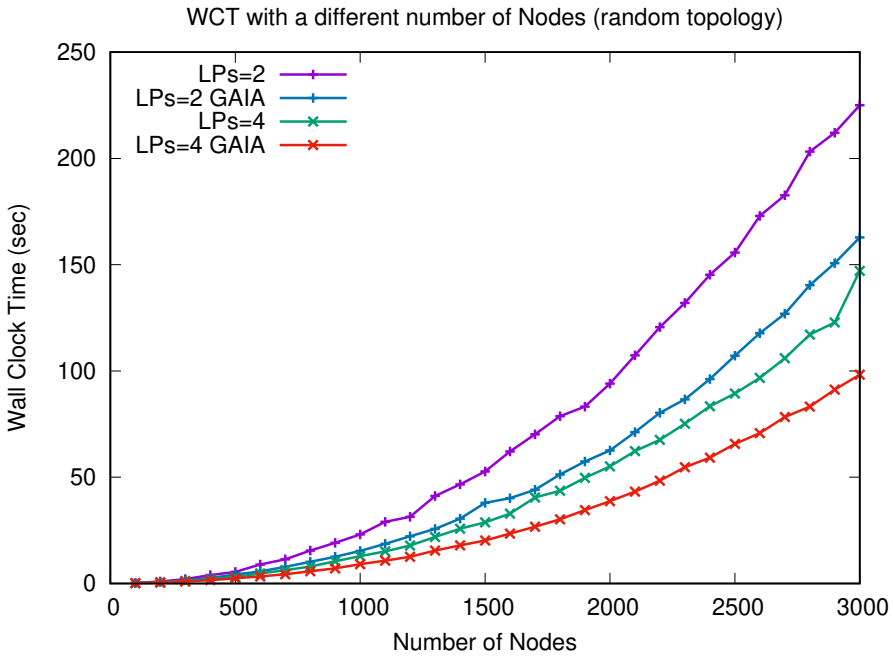

Figure 4: Wall-Clock-Time of a simulation run (random network topology) in parallel setups (i.e. 2-4 LPs) with an increasing number of Nodes. Lower is better.

straints of the PADS. The direct effect of breaking such a group of entities is a reduction of the system's ability to cluster the communication in the same LP. The cascading effect is a reduction of the amount of "cheap" intra-LP communication and an increase of "costly" communication overhead due to the inter-LPs communication (see Figure 7).

Figure 6 shows the speedup for all the configurations reported in the figures. The speedup is a metric typically used in parallel/distributed systems to compare the performance of a parallel/distributed setup with a sequential one that runs on a single CPU core. In this case, we have considered the parallel execution (i.e. GAIA OFF), the parallel execution with communication load balancing (i.e. GAIA ON), and the parallel execution with both communication and computational load-balancing (i.e. GAIA+). Since random networks have an average degree with a distribution that is quite regular, i.e. with limited imbalances among the nodes, we do not expect that GAIA+ is able to provide a significant speedup in the execution. In other words, GAIA+ is designed to react to imbalances in the simulation execution, but

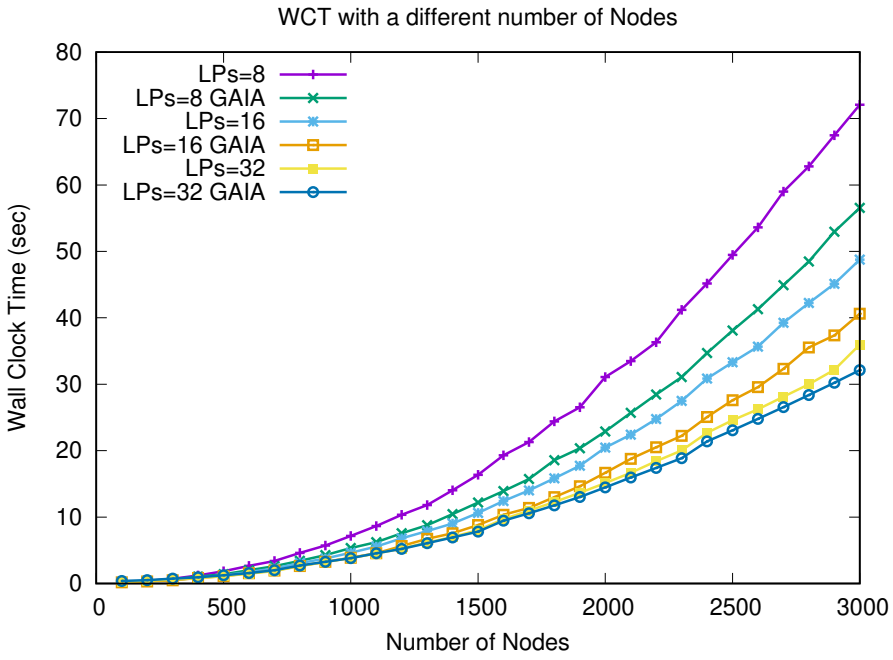

Figure 5: Wall-Clock-Time of a simulation run (random network topology) in parallel setups (i.e. 8-32 LPs) with an increasing number of Nodes. Lower is better.

both the simulation model and the execution architecture considered in this evaluation have no relevant imbalances. For this specific evaluation, the best speedup for the random topology simulation model with 3000 nodes is obtained with 30 LPs. With respect to the parallel approach (i.e. GAIA OFF), both GAIA and GAIA+ are able to provide a small speedup increase in many configurations.

Generally speaking, given the complexity of the simulation model, the very limited computation running on each node, and the huge amount of communication that is required by the gossip-based dissemination protocol, a speedup larger than 4 with 8 LPs can be considered a good outcome. In other words, the model considered in this paper is very communication-bound and therefore it is near to be a worst-case for PADS. Conversely, simulation of models that are more CPU-bound would obtain even larger speedup values. The effect of GAIA on the simulation model execution is significant but could be further increased if clustering mechanisms, specifically aimed at complex networks, are implemented in the simulation middleware. In fact,

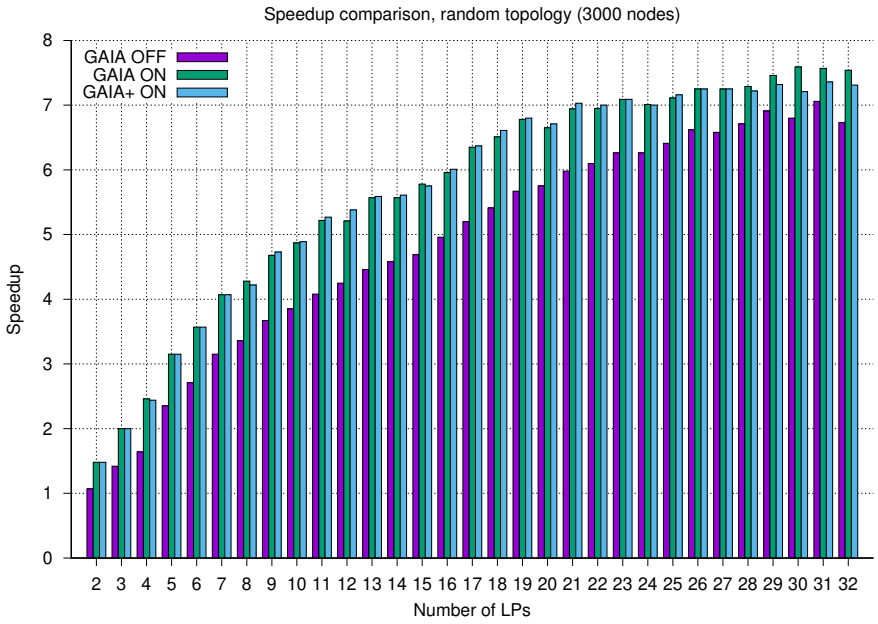

Figure 6: Speedups with a different number of LPs and load-balancing configurations. 3000 Nodes in a random topology. Higher is better.

in this performance evaluation, we have opted to use the generic clustering heuristics provided by GAIA without any form of specific adaptation and a very limited tuning. This decision was motivated by the fact that we are interested in obtaining results that can be easily generalized and we are not much interested in very specific setups that would give a limited contribution. Finally, GAIA+ is unable to provide a significant gain. As explained above, this was somehow expected given the specific characteristics of random networks, i.e. degree distribution and topology structure. On the other hand, it is interesting to note that the results show that the mechanism used for implementing GAIA+ does not introduce a significant overhead. This means that, in presence of a shared execution architecture, in which it can not be avoided the presence of possibly unexpected (computation or communication) background load, GAIA+ would be able to improve the simulation execution, i.e. changing the load of specific LPs to balance the background load caused by other running processes, without reducing the simulation execution speed. This feature could be very useful when running simulations on shared execution architectures.

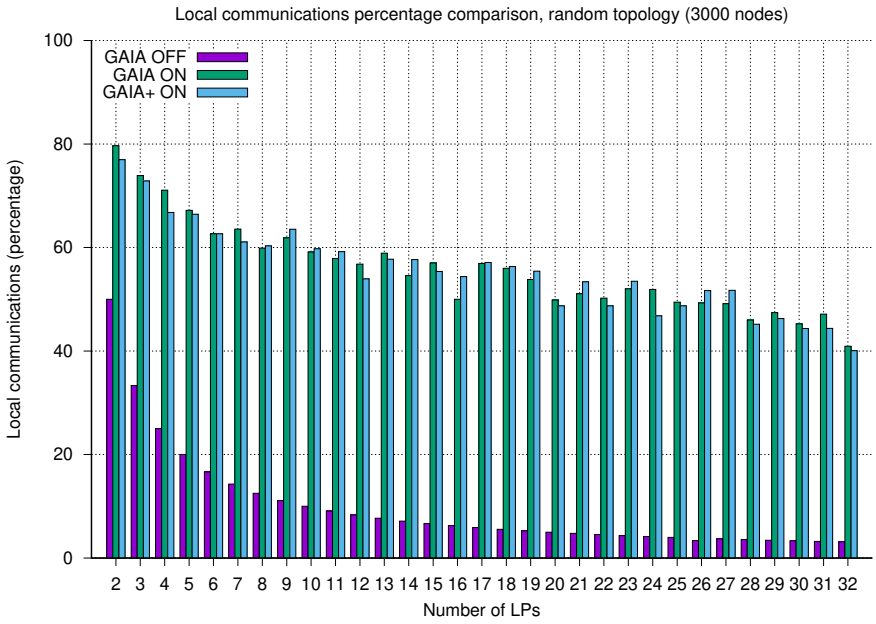

Figure 7: Local communication (i.e. intra-LP) percentage with an increasing number of LPs and different load-balancing configurations. 3000 Nodes in a random topology. Higher is better.

In Figure 7, we investigate the behavior of GAIA and GAIA+ in terms of their ability to cluster the intra-LP communications when in presence of an increasing number of LPs and a fixed number of simulated nodes (i.e. 3000). As shown in the figure, the absence of a load-balancing mechanism (i.e. GAIA OFF) reduces the amount of local communication while increasing the partitioning of the simulated model. This figure demonstrates that GAIA, in presence of a random graph topology, is able to effectively increase the percentage of local communications by clustering the interacting simulated nodes in the same LP. Clearly, the trend in the figure is declining with the increase of the number of LPs, since the problem is getting harder and harder. GAIA+ shows a similar behavior, but with a small reduction of its efficiency that is due to the further constraints imposed by the computational load-balancing mechanism.

5.0.2. Scale-free topology

In this part of the performance evaluation, we report the results of the same experiments seen above in the presence of networks generated with a

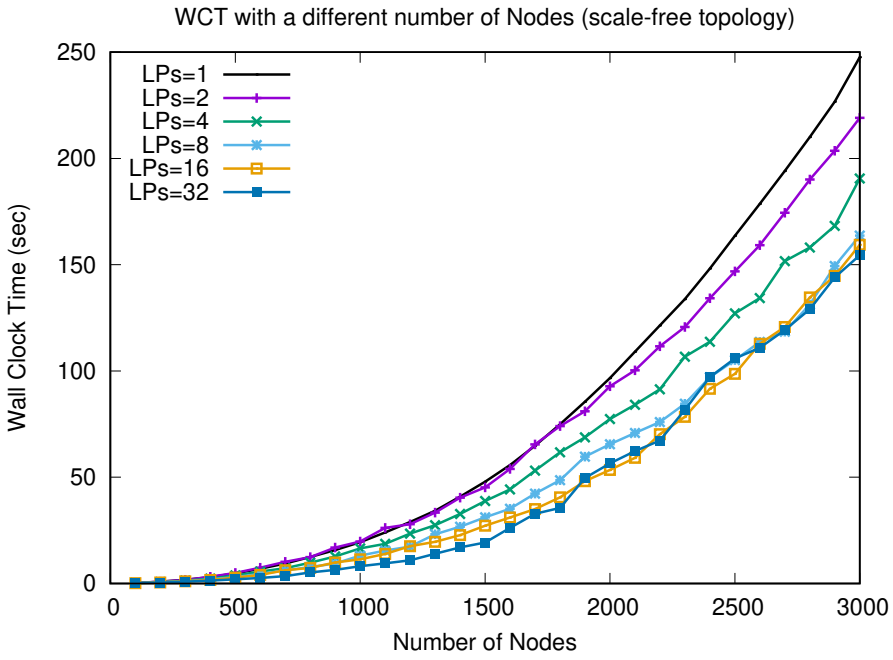

Figure 8: Wall-Clock-Time of a simulation run (scale-free network topology) in sequential and parallel setups (i.e. GAIA OFF) with an increasing number of Nodes. Lower is better.

scale-free topology. As shown in Figure 1, the network topology has a negligible impact on the number of messages that are delivered by the gossip-based dissemination protocol. Under the simulation viewpoint, the big difference between the random and the scale-free network topologies is the degree distribution of nodes. In a scale-free topology, the nodes have a degree distribution that follows a power-law. This means that with respect to a random network, in the simulation there are significant imbalances among the different nodes in terms of sent/received messages with clear consequences in both terms of communication and computation overhead. In a sequential simulation (i.e. $LPs = 1$) the imbalances described above have no consequences since there is a single execution unit that is responsible for running all the simulated nodes (i.e. LPs). In other words, an execution architecture that is composed of a single node has no issues with load-balancing, but it can be easily overloaded. This can be observed by comparing the results reported in Figures 2 and 8, in which the WCTs for running random and scale-free simulations are very close. This happens also when 2 LPs are used, but the

gap between the WCTs enlarges with more than 2 LPs. In other terms, in the simulation of a scale-free network, the parallel approach is still able to produce a speedup, but this is reduced with respect to a random topology due to the imbalances in the simulated model.

Figure 8 shows that in the simulation of scale-free networks, the setups with more than 8 LPs are unable to give a relevant performance boost. More specifically, the WCTs that are obtained by 16 and 32 LPs are quite unsatisfactory since by increasing the number of simulated nodes we obtain a WCT that is equal to 8 LPs. In other words, adding more nodes to the execution architecture there is no gain in terms of reduction of the amount of time that is necessary to obtain the simulation results.

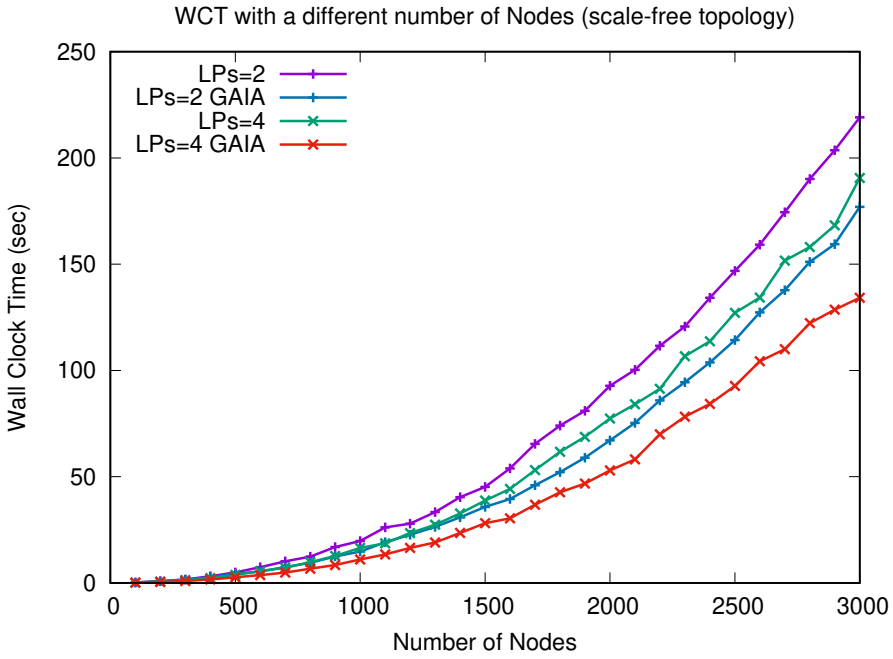

Figure 9: Wall-Clock-Time of a simulation run (scale-free network topology) in parallel setups (i.e. 2-4 LPs) with an increasing number of Nodes. Lower is better.

Figures 9 and 10 show that, also in this case, GAIA is able to provide a significant speedup of the execution. This means that, even if in the scale-free topology the communication among modes is very unbalanced, GAIA is able to self-cluster the nodes obtaining a satisfactory reduction in the communication cost (see also Figure 12). Moreover, it is worth noting that

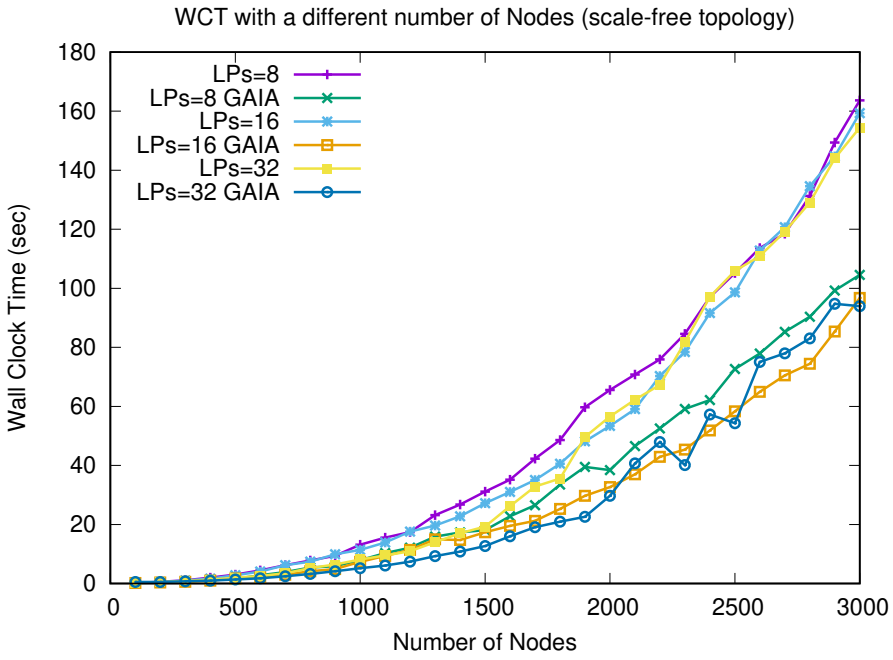

Figure 10: Wall-Clock-Time of a simulation run (scale-free network topology) in parallel setups (i.e. 8-32 LPs) with an increasing number of Nodes. Lower is better.

the setup with 2 LPs with GAIA ON is better than the setup with 4 LPs with GAIA disabled.

In terms of speedup, the best setup for the simulation of scale-free networks with 3000 nodes is obtained with 32 LPs with GAIA turned ON (see Figure11). There are two key aspects that are relevant. First, the performance of {8, 16, 32} LPs with GAIA ON are very close. Moreover, in the setup with 32 LPs, GAIA introduces a not negligible instability. In general, we can see that also in the case of scale-free topologies, with respect to the parallel approach (i.e. GAIA OFF), GAIA is still able to provide a relevant increase of the speedup. The gain obtained by GAIA in scale-free networks is higher with respect to random topologies. This demonstrates that in presence of significant imbalances in the communication load among nodes, the self-clustering mechanism is able to improve the allocation of the parallel simulation. On the other hand, GAIA+ does not introduce an overhead with respect to GAIA, but it fails in dealing with the imbalances that are caused by the scale-free networks' topology. In other words, GAIA+ is unable to

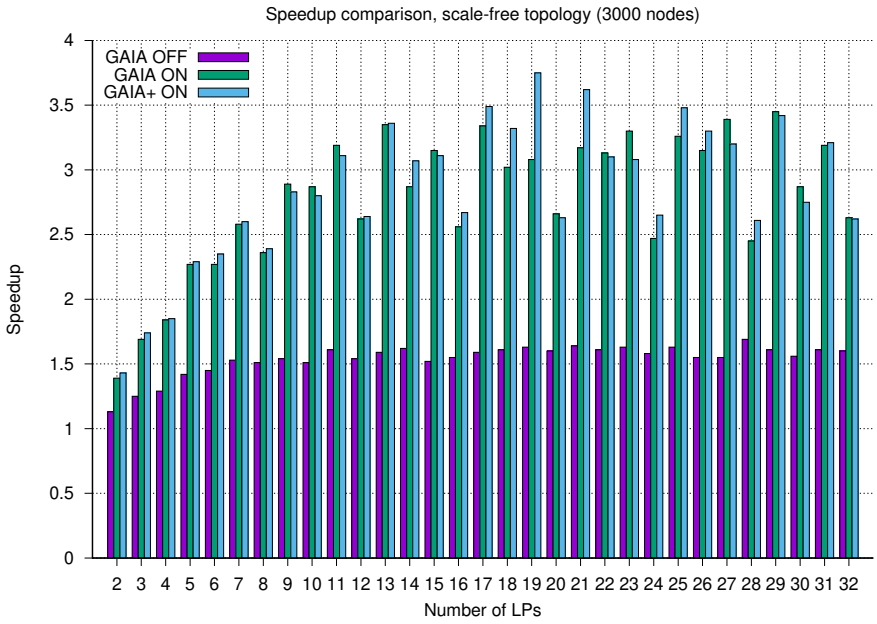

Figure 11: Speedups with a different number of LPs and load-balancing configurations. 3000 Nodes in a scale-free topology. Higher is better.

provide a speedup by migrating the simulated nodes, with the aim to obtain a more balanced execution in terms of computation among the LPs. This is due to both the characteristics of the load-balancing mechanism and the topology of the scale-free networks. More in detail, even if the degree distribution in a scale-free network is not uniform, i.e. there are few nodes with many links and many nodes with few links, the magnitude of this imbalance is reflected more in the communication cost than in the amount of computation to be executed on each simulated node. GAIA+, as seen above, is composed of a first part (i.e. GAIA) that clusters the highly interacting simulated entities in the same LP, i.e. communication load-balancing. Then, a computational load-balancing mechanism that tries to balance the work speed of the different LPs. By looking at the results, we can conclude that in the scale-free scenario described above, the GAIA+ mechanism is mainly dedicated to reducing the communication cost rather than being really able to obtain a smooth computation among the different LPs. In practice, the current implementation of GAIA+ considers as more important to reduce the communication cost than balancing the computation among the LPs.

Also in this case, simulation models of complex networks with more computational activity to be run in each node would have a more relevant benefit from GAIA+ with respect to the simulation model used in this performance evaluation.

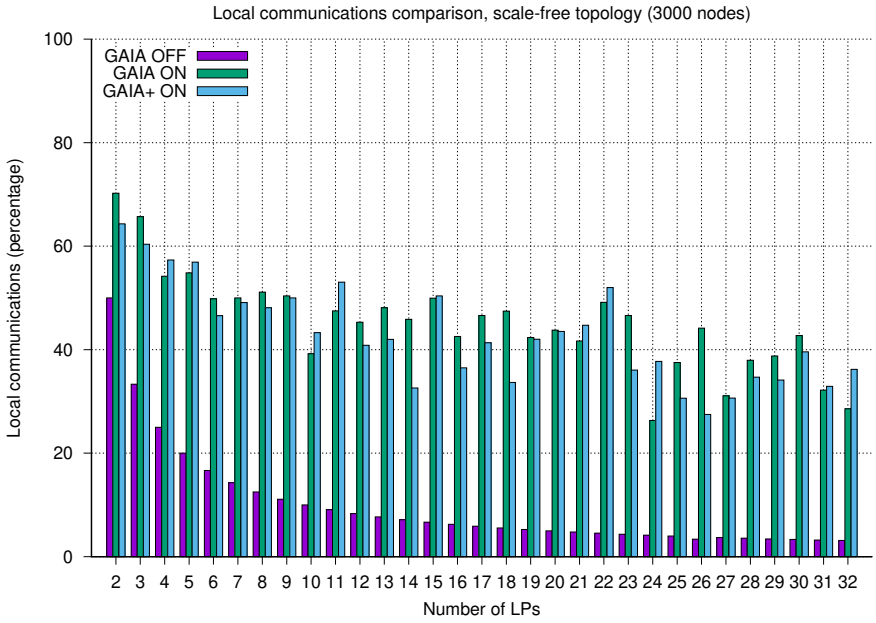

Figure 12: Local communication (i.e. intra-LP) percentage with an increasing number of LPs and different load-balancing configurations. 3000 Nodes in a scale-free topology. Higher is better.

In Figure 12, we show the behavior of GAIA and GAIA+ in terms of their ability to cluster the intra-LP communications when in presence of an increasing number of LPs and a fixed number of simulated nodes (i.e. 3000) organized in a scale-free topology. This figure demonstrates that GAIA, also in this case, is able to effectively increase the percentage of local communications but with respect to the random topology, the quality of clustering is significantly lower. In fact, the obtained percentage of local communications is lower with respect to the corresponding setup in a random topology. As described above, this is due to the uneven characteristics of scale-free topologies and of the communication on top of them. Again, GAIA+ shows a behavior that is similar to GAIA but now with a relevant instability of performances.

5.0.3. Random Graph vs Scale-Free Network Simulation

It is interesting to observe how the WCTs for simulating random graphs and scale-free networks differ. In particular, the WCT observed for the sequential execution for random and scale-free nets (i.e. LP=1, Figures 2 and 8) is similar. When we consider the PADS execution, without GAIA (Figures 4–5 and 9–10), the scale-free scenario requires a higher WCT than random networks. This is due to the fact that, even if the amount of generated messages is similar in both scenarios, the scale free net is "more complex" and difficult to handle. Hubs have many connections, resulting in a higher amount of messages going to different nodes. This results in a higher amount of inter-LP messages. When we activate GAIA, the clustering mechanism allows us to reduce this workload, thus reducing the WCT needed to perform the simulations.

6. Conclusions

In this paper, we have discussed the main characteristics in order to build effective simulation tools for large scale complex networks. We specifically focused on DES, with specific attention to the benefits introduced by exploiting a PADS approach. Our claim is that, through PADS, it is possible to simulate application protocols on top of complex network topologies. This allows us to simulate very diverse real systems and understand their behavior. This methodology fosters what-if analyses and enables experiments with a synthetic model of a system, when subject to different environmental conditions. This is not possible if we consider a simple analysis of static complex networks, through complex network theory. On the other hand, the use of traditional ABMS systems allows the creation of simple toy models, where the behavior of each agent is usually over-simplified, as well as the agent interactions. This is due to the fact that sequential simulators do not allow to scale both in terms of complexity of the model and size of the simulated system. We claim that the use of agent-based simulation over PADS methodologies, equipped with a tool able to generate and handle complex network topologies, allows taking the best of the two worlds, also increasing the scalability of the simulation. However, to the best of our knowledge, there were no current tools available in the literature.

We demonstrated our claims by evaluating LUNES, a parallel and distributed simulator we built. Thank to PADS approaches and adaptive migration capabilities, the tool is able to simulate large scale networks and mim-

icking application protocols on top of them, using an agent-based paradigm. The tool assigns simulated entities to different nodes in the simulation execution architecture e.g. CPU cores or hosts, and based on the level of interactions among entities, it clusters most interacting entities into the same Logical Process. This saves a lot of communication in the execution architecture, thus gaining speed in the execution of the simulation and scalability.

In the experimental evaluation, we generated networks of the order of thousands of nodes and then we varied the network topologies between random graphs and scale-free networks. These two network topologies are very different and thus represent two main benchmarks to study the behavior of the simulators. Results show that PADS strategies can be quite effective when they are employed with clever adaptive migration strategies, which are able to cluster interacting simulator entities, in order to reduce the communication overheads between different execution units.

Even if in this paper we have considered networks that are static in their topology, the proposed approach seems to be well suited also for the simulation of dynamic topologies, which are also referred to as temporal networks. In fact, the adaptive load-balancing mechanism provided by the GAIA/GAIA+ framework should be able to quickly react to imbalances that are caused by changes in the network topology. For example, by adding or deleting new nodes and their edges. Clearly, the level of dynamicity of the temporal network will have to be carefully considered. In particular, in the design of the self-clustering heuristics used for trigger re-allocations in the parallel/distributed architecture. In any case, our results suggest that the framework we proposed in this paper represents a viable approach to investigate dynamicity aspects.

Acronyms

DES Discrete Event Simulation

HPC High Performance Computing

LP Logical Process

PADS Parallel And Distributed Simulation

PDES Parallel Discrete Event Simulation

PEU Physical Execution Unit

SE Simulated Entity

WCT Wall Clock Time

References

[1] igraph – the network analysis package, http://http://igraph.org, 2022.

[2] MASON: A new multi-agent simulation toolkit, `https://cs.gmu.edu/~eclab/projects/mason/`, 2022.

[3] Parallel And Distributed Simulation (PADS) Research Group, `http://pads.cs.unibo.it`, 2022.

[4] R. Albert, A.L. Barabási, Statistical mechanics of complex networks, Rev. Mod. Phys. 74 (2002) 47–97.

[5] R. Alkharboush, R.E. De Grande, A. Boukerche, Load prediction in hla-based distributed simulation using holt's variants, in: 2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications, pp. 161–168.

[6] S. Aridhi, A. Montresor, Y. Velegrakis, Bladyg: A graph processing framework for large dynamic graphs, Big Data Research 9 (2017) 9 – 17.

[7] H. Bao, J. Bielak, O. Ghattas, L. Kallivokas, D. O'Hallaron, J. Shewchuk, J. Xu, Large-scale simulation of elastic wave propagation in heterogeneous media on parallel computers, Computer Methods in Applied Mechanics and Engineering 152 (1998) 85–102.

[8] K. Bochenina, S. Kesarev, A. Boukhanovsky, Scalable parallel simulation of dynamical processes on large stochastic kronecker graphs, Future Generation Computer Systems 78 (2018) 502 – 515.

[9] S. Bosmans, T. Bogaerts, W. Casteels, S. Mercelis, J. Denil, P. Hellinckx, Adaptivity in distributed agent-based simulation: A generic load-balancing approach, in: S. Swarup, B.T.R. Savarimuthu (Eds.), Multi-Agent-Based Simulation XXI, Springer International Publishing, Cham, 2021, pp. 1–12.

[10] S. Boyd, A. Ghosh, B. Prabhakar, D. Shah, Randomized gossip algorithms, IEEE/ACM Trans. Netw. 14 (2006) 2508–2530.

[11] K. Chandy, J. Misra, Asynchronous distributed simulation via a sequence of parallel computations, Communications of the ACM 24 (1981) 198–206.

[12] 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.

[13] G. Cordasco, V. Scarano, C. Spagnuolo, Distributed mason: A scalable distributed multi-agent simulation environment, Simulation Modelling Practice and Theory 89 (2018) 15–34.

[14] R.E. De Grande, A. Boukerche, Dynamic partitioning of distributed virtual simulations for reducing communication load, in: 2009 IEEE International Workshop on Haptic Audio visual Environments and Games, pp. 176–181.

[15] S. Ding, C. Chen, B. Xin, P.M. Pardalos, A bi-objective load balancing model in a distributed simulation system using nsga-ii and mopso approaches, Applied Soft Computing 63 (2018) 249–267.

[16] R. Dobrescu, S. Taralunga, S. Mocanu, Web traffic simulation with scale-free network models, in: AIC'07: Proc. of the 7th Conference on 7th WSEAS International Conference on Applied Informatics and Communications, WSEAS, 2007.

[17] G. D'Angelo, The simulation model partitioning problem: an adaptive solution based on self-clustering, Simulation Modelling Practice and Theory (SIMPAT) 70 (2017) 1 – 20.

[18] G. D'Angelo, S. Ferretti, Highly intensive data dissemination in complex networks, Journal of Parallel and Distributed Computing 99 (2017) 28 – 50.

[19] E. Egea-Lopez, J. Vales-Alonso, A. Martinez-Sala, P. Pavon-Mario, J. Garcia-Haro, Simulation scalability issues in wireless sensor networks, Communications Magazine, IEEE 44 (2006) 64 – 73.

[20] G.S. Fishman, Discrete-Event Simulation, Springer-Verlag, Berlin, Heidelberg, 2001.

[21] R. Fujimoto, Parallel discrete event simulation, Communications of the ACM 33 (1990) 30–53.

[22] R. Fujimoto, Parallel and Distributed Simulation Systems, *Wiley & Sons*, 2000.

[23] R.M. Fujimoto, Parallel discrete event simulation, in: Proceedings of the 21st conference on Winter simulation, WSC '89, ACM, New York, NY, USA, 1989, pp. 19–28.

[24] B. Garbinato, D. Rochat, M. Tomassini, Impact of scale-free topologies on gossiping in ad hoc networks., in: NCA, IEEE Computer Society, 2007, pp. 269–272.

[25] J.E. Gonzalez, Y. Low, H. Gu, D. Bickson, C. Guestrin, Powergraph: Distributed graph-parallel computation on natural graphs, in: Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation, OSDI'12, USENIX Association, Berkeley, CA, USA, 2012, pp. 17–30.

[26] V. Grimm, E. Revilla, U. Berger, F. Jeltsch, W. Mooij, S. Railsback, H.H. Thulke, J. Weiner, T. Wiegand, D. DeAngelis, Pattern-oriented modeling of agent-based complex systems: Lessons from ecology, Science 310 (2005) 987–991.

[27] B. Hou, Y. Yao, B. Wang, D. Liao, Modeling and simulation of large-scale social networks using parallel discrete event simulation, SIMULATION 89 (2013) 1173–1183.

[28] T. Hruz, S. Geisseler, M. Schöngens, Parallelism in simulation and modeling of scale-free complex networks, Parallel Computing 36 (2010) 469–485.

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

[30] M. Jelasity, A. Montresor, O. Babaoglu, Gossip-based aggregation in large dynamic networks, ACM Trans. Comput. Syst. 23 (2005) 219–252.

[31] T.G. Kannampallil, G.F. Schauer, T. Cohen, V.L. Patel, Considering complexity in healthcare systems, Journal of Biomedical Informatics 44 (2011) 943 – 947.

[32] Z. Khayyat, K. Awara, A. Alonazi, H. Jamjoom, D. Williams, P. Kalnis, Mizan: A system for dynamic load balancing in large-scale graph processing, Proceedings of the 8th ACM European Conference on Computer Systems, EuroSys 2013, pp. 169–182.

[33] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Commun. ACM 21 (1978) 558–565.

[34] C.M. Macal, M.J. North, Agent-based modeling and simulation, in: Winter Simulation Conference, WSC '09, Winter Simulation Conference, 2009, pp. 86–98.

[35] G. Malewicz, M.H. Austern, A.J. Bik, J.C. Dehnert, I. Horn, N. Leiser, G. Czajkowski, Pregel: A system for large-scale graph processing, in: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, SIGMOD '10, ACM, New York, NY, USA, 2010, pp. 135–146.

[36] Z. Miao, P. Yong, Y. Mei, Y. Quanjun, X. Xu, A discrete pso-based static load balancing algorithm for distributed simulations in a cloud environment, Future Generation Computer Systems 115 (2021) 497–516.

[37] R.D. Michele, S. Ferretti, M. Furini, On helping broadcasters to promote tv-shows through hashtags, Multimedia Tools Appl. 78 (2019) 3279–3296.

[38] N. Minar, R. Burkhart, C. Langton, M. Askenazi, et al., The swarm simulation system: A toolkit for building multi-agent simulations (1996).

[39] A. Montresor, M. Jelasity, PeerSim: A scalable P2P simulator, in: Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09), Seattle, WA, pp. 99–100.

[40] M. Newman, Networks: An Introduction, Oxford University Press, Inc., New York, NY, USA, 2010.

[41] V.T. Nguyen, R. Fujimoto, Link partitioning in parallel simulation of scale-free networks, in: Proceedings of the 20th International Symposium on Distributed Simulation and Real-Time Applications, DS-RT '16, IEEE Press, Piscataway, NJ, USA, 2016, pp. 77–84.

[42] M. North, N. Collier, J. Vos, Experiences creating three implementations of the repast agent modeling toolkit, ACM Transactions on Modeling and Computer Simulation 16 (2006) 1–25.

[43] R.S. Pienta, R.M. Fujimoto, On the parallel simulation of scale-free networks, in: Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, SIGSIM PADS '13, ACM, New York, NY, USA, 2013, pp. 179–188.

[44] H. Sarjoughian, B. Zeigler, S. Hall, A layered modeling and simulation architecture for agent-based system development, Proceedings of the IEEE 89 (2001) 201–213.

[45] S. Verma, W.T. Ooi, Controlling gossip protocol infection pattern using adaptive fanout, in: ICDCS '05: Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, IEEE Computer Society, Washington, DC, USA, 2005, pp. 665–674.

[46] U. Wilensky, NetLogo, http://ccl.northwestern.edu/netlogo/, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL, 1999.

[47] K. Yocum, E. Eade, J. Degesys, D. Becker, J. Chase, A. Vahdat, Toward scaling network emulation using topology partitioning, in: 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003., pp. 242–245.

[48] G. Zacharewicz, C. Frydman, N. Giambiasi, G-DEVS/HLA environment for distributed simulations of workflows, SIMULATION 84 (2008) 197–213.

[49] B.P. Zeigler, G. Ball, H. Cho, J. Lee, H. Sarjoughian, Implementation of the DEVS formalism over the HLA/RTI: Problems and solutions, in: Proceedings of the Simulation Interoperation Workshop (SIW), 1999.

[50] B.P. Zeigler, A. Muzy, E. Kofman, Theory of Modeling and Simulation: Discrete Event and Iterative System Computational Foundations, Academic Press, Inc., USA, 3rd edition, 2018.

[51] X. Zeng, R. Bagrodia, M. Gerla, Glomosim: a library for parallel simulation of large-scale wireless networks, SIGSIM Simul. Dig. 28 (1998) 154–161.