Dottorato di Ricerca in Informatica

Università di Bologna e Padova

ARTÌS: Design and Implementation of an Adaptive Middleware for Parallel and Distributed Simulation

Gabriele D'Angelo

March 2005

Coordinatore:

Prof. Ozalp Babaoglu

Tutore:

Prof. Lorenzo Donatiello

Abstract

The simulation is a technique of primary importance in the design and verification of systems and architectures. The parallel and distributed simulation has been proved to reduce the time required to complete the simulation of some scenarios, improve the code reusability, address the requests for fault-tolerance and support the spatially located architectures. In the following of this thesis it will be presented a new simulation middleware named Advanced RTI System (ARTÌS), designed to support the parallel and distributed simulations of complex systems characterized by heterogeneous and distributed model components. The runtime logical structure and the implementation details will be described, some scenarios considered of interest (i.e. ad hoc and sensor wireless networks) have been simulated as testbed evaluation for the proposed middleware. Our effort to improve the simulation speed-up of parallel and distributed simulation with respect to a sequential monolithic approach has involved new features as the "simulated entities migration" and the "concurrent replication" of simulation runs. Finally some concepts derived by our work with ARTÌS will be applied to the design and implementation of a new middleware for massively populated Internet Games.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

During last years, simulation has step by step gained an increased importance in design and verification of systems and architectures. Interesting systems are often complex but, mainly in the past, their evaluation was conditioned by inadequate computational capabilities. Sometimes the resulting approach was an oversimplified conceptual model: a first step in direction of partial or wrong results. Hence, tools used to build simulators are of primary importance in the design and implementation of complex systems.

The simulation oriented analysis of some systems is well fitted for an approach based on Parallel And Distributed Simulation (PADS). The traditional sequential monolithic approach is often inadequate to complete the simulation execution. The monolithic approach limitations often are due to limited resources (i.e. system memory) or to the excessive amount of time necessary to complete the runs. As remarked in [48, 49] the distributed computation is not always favorable in terms of performance. Highly interactive dynamic environments are difficult to simulate on a distributed architecture, often the generic solution is based on oversimplified interaction filtering mechanisms. Defining the terminology, it will referred as "parallel simulation" each architecture where two ore more Physical Execution Units (PEUs) are interconnected by a low latency communication bus (i.e. SMP shared memory

systems). Conversely it will referred as "distributed simulation" each architecture based on a set of loosely-coupled PEUs, interconnected by a high latency network (i.e. LAN, WAN, Internet).

The PADS techniques obviously have drawbacks, the load sharing is not always easy or feasible, the load balancing between the set of PEUs running the simulation is usually a difficult to solve problem. The speed-up obtained by load balancing can be worn away by the communication overhead necessary to preserve synchronization in the simulation cluster. Traditionally, a big amount of research work has been done in the field of synchronization and data distribution management (DDM) to reduce the communication overheads, increasing the PADS speed-up with respect to the monolithic approach.

In the real world, the exploiting of PADS techniques is not a news, a large set of communities has shared knowledge and experiences directly or indirectly related to the parallel and distributed simulation field. Hence, during the last decades, a lot of simulation tools has been developed and deployed: some tools are proprietary, other are in some way more "open". In 2000 the need for a common definition for distributed simulation has been satisfied with the High Level Architecture (IEEE 1516) standard approval.

1.1 Problem statement

The definition of a well designed standard is a grow and unification factor for the distributed simulation field, but a lot of runtime middleware implementations (RTI) standard compliant have proved to have unsatisfactory performance.

Some scenarios considered of high relevance are extremely dynamic and massively populated (i.e. ad hoc and sensor wireless networks). This kind of simulations is uneasy to run efficiently in a distributed environment based on traditional data filtering techniques. The communication overhead caused by the simulation of massively dynamic environment would nullify the gain obtained by the load distribution.

The wireless networks topology is really dynamic with respect of time evolution and, static pre-allocation would become sub-optimal in a few timesteps.

The actual IEEE 1516 standard definition is totally missing the "simulated entities migration" paradigm, that in our opinion could be valuable to improve the overall simulation speedup. Moreover the standard is really complex to implement, to learn and to use and so unsuited for environments requiring a simpler approach.

1.2 Thesis contributions

In order to overcome these problems, we analyzed the current situation of the commonly used runtime implementations and we proposed a new framework named Generic Adaptive Interaction Architecture (GAIA) devoted to externally introduce "simulated entities migration" capabilities to an unmodified release of the DMSO HLA runtime [49].

The evolution of our work described in [48, 49] and of the testbed experiences in [48] are the main topic of this thesis. Our research work in the field of parallel and distributed simulation was started with at least two main goals: good performances and improved usability. The most important side-effect of our research is the design and implementation of a new simulation middleware named Advanced RTI System (ARTÌS), designed to support parallel and distributed simulations of complex systems characterized by heterogeneous and distributed model components [45]. The ARTÌS design is oriented to support the model components heterogeneity, distribution and reuse, and to increase the simulation performances, scalability and speed-up, in parallel and distributed simulation scenarios.

The ARTÌS middleware has been the basis for an extensive work on performances, involving dynamic adaptation of the interprocess communication layer and concurrent replication of parallel and distributed simulation. Another topic of further research was the "simulated entities migration" paradigm, in this case directly applied within the ARTÌS middleware, obtaining an integration of the GAIA framework and our new middleware [47].

The aim of this thesis is to discuss our researches in the field of parallel and distributed simulation. To describe the design and implementation of ARTÌS and to analyze the performance results obtained during the simulation of some interesting wireless networks (i.e. ad hoc and sensor) [46]. This thesis will also describe how some paradigms as "simulated entities migration" and "concurrent replication of runs" can be valuable to achieve a better speed-up in the simulation execution.

Finally we will see as Internet Gaming architectures and PADS are quite related to share some important concepts and techniques. Taking inspiration from our simulation-related work we have been able to introduce an improved architecture to support massively populated Internet Games.

1.3 Outline

The rest of this dissertation is organized as follows. Chapter 2 recalls some background concepts about Parallel Discrete Event Simulation (PDES). This chapter is intended as an introduction to the basic notions needed in the rest of the thesis. Chapter 3 describes the motivations, the internal logical architecture and implementation of ARTÌS, a new middleware for parallel and distributed simulation. Chapter 4 contains some examples of how ARTÌS can be used to implement real simulations of large scale wireless systems (ad hoc and sensors networks). Chapter 5 introduces the Concurrent Replication of Parallel and Distributed Simulations (CR-PADS) approach to reduce the Wall-Clock-Time (WCT) necessary to complete simulations. Chapter 6, starts from the distributed simulation foundations and shows that a class of Internet games can be seen as a kind of simulations where some constraints could be relaxed. The chapter proposes a new middleware architecture to support massively populated Internet Games. The concluding chapter summarizes the main results of this thesis, and discusses the future work.

Chapter 2

Background

> *"There are more things in heaven and earth, Horatio, than are dreamt of in your philosophy."*
> William Shakespeare, Hamlet

A simulation is a system that represents or emulates the behavior of another system over time. In a computer simulation the system doing the emulating is a computer program [78]. In a very simplified form a computer simulation can be seen as a set of evolving state variables. The evolution of state variables over time can be managed with different timings. Time flow mechanism are continuous or discrete, in this thesis we are interested in the following: state variables will be updated at discrete points in the simulation time. The simulation advance of time can be time-stepped or event-driven, in the first case the simulation state is updated every a fixed amount of time (Figure 2.1), in the last only in presence of events (a "change" happens in the simulation) the state variables are updated (Figure 2.2).

2.1 Parallel and distributed simulation

The state variables evolution and the events computation can be managed in a sequential monolithic approach or in a decentralized way. In the last case we will refer to parallel or distributed simulation. If the simulation is executed by a set of Physical Execution Units (PEU) interconnected by a low latency network (i.e.

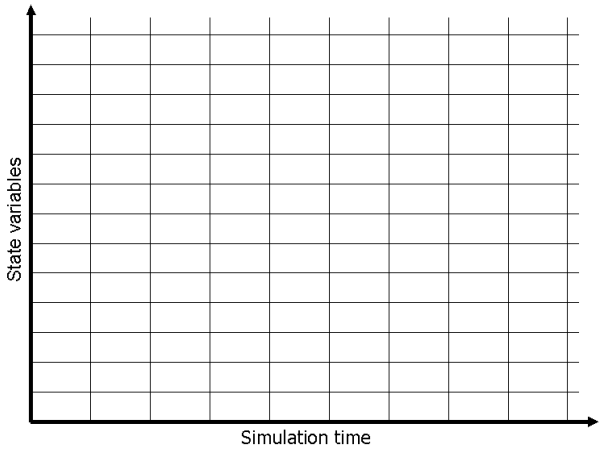

Figure 2.1: Time-stepped approach

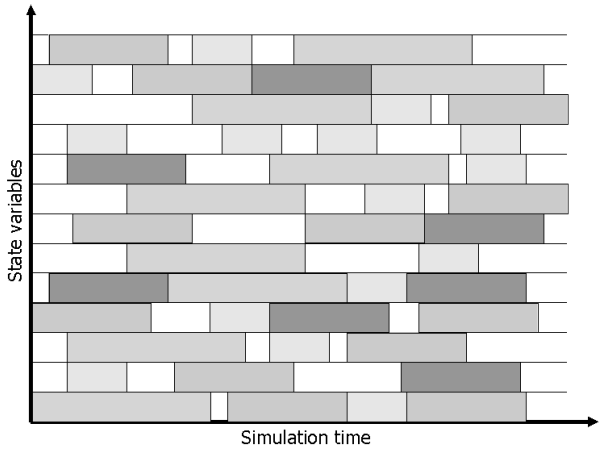

Figure 2.2: Event-driven approach

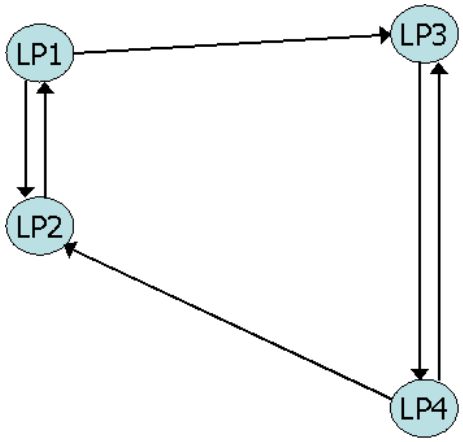

Figure 2.3: A set of interacting Logical Processes (LPs)

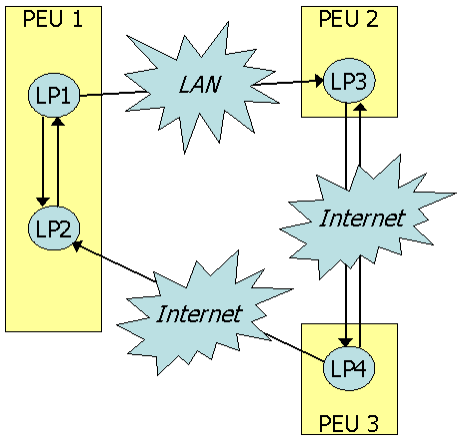

Figure 2.4: A set of interacting Physical Execution Units (PEUs)

shared memory multiprocessor), we will refer to a parallel simulation. Conversely in presence of a high latency network (i.e. LAN, WAN, Internet) we will refer to a distributed simulation. The physical system can be viewed as a collection of physical processes (executed on a set of PEUs) that interact in some fashion with each physical process being modeled by a logical process (LP) [78]. Each LP is usually executed by a single PEU, the interactions are performed exchanging time stamped messages.

2.2 Synchronization algorithms

In a centralized system the notion of time is strictly defined, generated events can be easily processed following a total ordering thanks to the CPU clock, assumed as main reference of timing. A distributed environment has not a single reference clock but a pool of CPU clocks subjected to skews and different speeds. Without defining any notion of ordering and synchronization the simulation semantics would be different depending if the simulation is executed on a single processor environment or a distributed system, it is need a set of constraints to preserve the original semantics. In a distributed architecture two event can be in a dependency order while processed on different CPUs, Lamport [95] has introduced the "happens before" relationship that is defined as follows:

- if E1 and E2 were generated in the same logical process and in sequence E1 is before E2 then "E1 happens before E2";
- if E1 is the event of sending a message from LP A to LP B and E2 is the receiving event then "E1 happens before E2" (in other words sending a message is time consuming);
- the "happens before" relationship is transitive.

The Lamport's "happen before" relationship defines a partial ordering in the set of events. In this environment two events are "casually ordered" if the Lamport's relationship is true, else they are "concurrent events". It important to note that "concurrent events" can be processed in parallel and "causally ordered" events have to be processed sequentially, introducing a form of synchronization between the distributed processing units involved in the whole simulation. The defined partial ordering between events permits to improve the simulation speed processing in parallel the events that are not involved in a causally ordered relationship. Traditionally a great amount of research has involved the synchronization algorithms, it is required the guarantee that in centralized and distributed environments it will be obtained exactly the same simulation results, but this goal can be achieved in

radically different manners. The main approaches to this problems are two: conservative (pessimistic) and optimistic algorithms, based on opposite assumptions. In following I will sketch both approaches, for a complete reference, the work referred in [108] represents a dated but still valuable survey about synchronization and related topics.

2.2.1 Conservative approach

The most widely used conservative synchronization algorithm is the "Chandy-Misra-Bryant" (CMB). The overall simulation is decomposed in a set of units called Logical Process (LP), each LP manages the evolution of one or more simulated entities. The communication between simulated entities occurs through message passing, the timestamped messages are sent over reliable communication channels. The set of defined channels is static with respect of the simulation evolution, each channel interconnects a pair of LPs and it is not only reliable but also preserves the message order.

The CMB algorithm needs to determine "safe" events, an event is defined as safe when its computation will not bring to violations in the local causality constraint. In a conservative algorithm a processor is not permitted to execute an event until it is certain to not receive a message in its past [108]. A logical process can verify this condition if and only if each incoming channel (queue) contains at least one message. The messages arriving in each receiving channel are enqueued in timestamp order, and so it is easy to verify that the message with the minimum timestap is safe and can be immediately processed. If during the simulation execution a channel queue becomes empty then the algorithm can not determine what is the minimum timestamped message that could be received from the empty channel and obviously can not calculate a global minimum. In absence of safe event to process, the LP has to wait until all channels contains at least one message. This waiting condition can produce a LPs circular wait and a deadlock situation where all the LPs are starving, waiting for new messages. The NULL message is a special message that usually does not contain any model-related data, it is useful only to support the simulated time

advancement. Every time a LP advances its simulated time, or a new outbound message is sent, the LP generates also a set of NULL messages sent to all neighbors.

Thanks to NULL messages, the logical process is able to compute a new lower bound on messages that could be received from neighbors, advancing the computation. Introducing the NULL messages has permitted to break the circular waiting but the overhall number of sent messages was greatly increased. Thanks to this version of the conservative algorithm, the empty queues problem has been solved but the algorithm is still prone to livelock: all LPs could be exchanging timestamped NULL messages but the simulation time at each round could remain unchanged. To solve this further problem it was introduced the lookahead value, in [78] it is defined as: "if a logical process at simulation time T can only schedule new events with time stamp of at least $T+L$, then L is referred to as the lookahead for the logical process". Thanks to the lookahead value and NULL messages combination it is possible to break this livelock, in each round the simulated time will be for certain incremented. The legacy CMB algorithm is unable to simulate systems where the lookahead is equal to zero. Another main drawback of this algorithm is the big influence of the lookahead value over simulation performances, small lookahead values imply a large number of NULL messages sent on the communication network, increased overhead and general communication latency. Another drawback of this approach is due to the lookahead semantic, determining its value is strictly dependent by the simulated model and can not be generalized.

2.2.2 Optimistic approach

In the optimistic approach determining if an event is safe is no more necessary. The first proposed and most diffused optimistic synchronization algorithm is called Time Warp and has been proposed by Jefferson [91]. In the local control mechanism of this algorithm each LP executes all the enqueued events in timestamp order but without any "safe check": in absence of any form of synchronization the arrival of a message with a timestamp in the past (called straggler message) can happen. The optimistic approach is based on the assumption that causality violations are quite

rare and that is better to manage the violations consequences instead of paying extra overhead to avoid them.

To repair the causality violations it is necessary to roll-back the LP state variables, this operation does not only involve undoing modifications to the local state variables but also unsending messages. The old computation performed by the LP, now based on wrong state and arrived messages, may have produced a set of further messages spread all over the network to other LPs. To rollback not only the local LP state but also the whole simulation it is necessary to "unsend" such wrong messages and this is managed by anti-messages. The payload of anti-messages is the same of the corresponding positive message but with opposite mark, anti-messages are used to annihilate still unprocessed positive messages or to cause the roll-back of other LPs. Often the arrival of a straggler message determines a storm of anti-messages and state roll-backs all over the distributed simulation. This domino effect will propagate only into the simulated time future (starting from the straggler event timestamp) and so it will not happen a whole simulation cancellation. When the storm is over, all the simulation is now back to a consistent state and the LPs can start again to process events in timestamp order and send new messages.

The described algorithm is quite simple but its implementation requires some data structures to efficiently manage roll-backs. Without a global control mechanism this structures will continue to grow until the simulation end, it's required a further algorithm to determine if historical information can be discarded and reclaimed (fossil collection). The easier approach to fossil collection is to determine a lower bound on the time downgrade that can be caused by roll-backs: this lower bound is usually called Global Virtual Time (GVT). The GVT is defined as "the minimum time stamp among all unprocessed and partially processed messages and anti-messages in the system at wallclock time T" [78]. After calculating the GVT, each LP will be able to discard all the data structures related to information timestamped before the GVT value, and now unnecessary. The GVT computation can be managed by different centralized or distributed algorithms (i.e. Samadi's and Mattern's), the performance of different approaches is strictly dependent on the

simulation architecture. A large amount of research has involved optimizing the Time Warp algorithm, some mechanisms as dynamic memory allocation vs. static preallocation, infrequent state saving vs. copy state method vs. incremental state saving, lazy cancellation and re-evaluation, has been proposed, often with valuable speed-up achievements but it is worth to note that all optimizations and also the main algorithm is quite sensible to the simulation architecture and the simulated model: small modifications in the above factors can led to greatly influenced speed-up results.

2.3 Data Distribution Management (DDM)

A distributed simulation can be composed by thousands of simulated entities managed by a set of PEUs, interconnected by different kinds of networks. Each entity can be an information producer, consumer or event both. It is needed a mechanism to match consumers and producers, both are executed in a distributed environment by different PEUs. The simulated entities should be able to express interest to publish or receive some kind of information, these expressions of interest obviously have to be dynamic with respect of the time evolution. Following the requests phase and determined the correct matching, an optimized information spreading would require to exactly determine what data set has to be sent to each PEUs, minimizing the amount of data sent and taking advantage of eventual information duplicates. The Data Distribution Management (DDM) as defined in the High Level Architecture (HLA) (Section 2.5) is based on Class-Based Data Distribution and Routing Spaces ([78]:247 – 256). As we have demonstrated in [49] this approach could be not optimal for parallel and distributed simulation of wireless mobile systems and dynamically interacting systems, and so we have proposed a migration based approach (Chapter 4). The implementation of DDM services should be efficient and scalable, it can be based on different algorithms and mechanisms. In [55] is proposed a new approach to multicast group allocation, based on Dynamic Grid-Based Allocation, a combination of a Fixed Grid-Based method and Sender-Based strategy. In Section 3.2 other

useful mechanisms are sketched.

2.4 Parallel and distributed simulation of wireless networks

The state of the art in large scale wireless network simulation includes a fairly large number of monolithic tools and only a limited set of parallel and distributed environments. Most existing network simulation packages have severe limitations on the size and complexity of the network being modeled [117]. Generally speaking, monolithic tools usually are subjected to a very limited amount of available memory and poor computation performances. Parallel and distributed tools main weakness can be related to the synchronization and data distribution management overhead. An usual approach to solve this problem is to reduce the simulated network size, looking for a way to insure that certain invariants remain constant despite the size reduction [117].

2.4.1 MAISIE

Maisie is a C-based simulation language that can be used for sequential and parallel execution of discrete-event simulation models [15, 38]. On top of MAISIE at UCLA has been implemented an advanced simulation environment which is used to examine, validate, and predict the performance of mobile wireless network systems [120]. Currently MAISIE is no longer supported by the UCLA Parallel Computing Laboratory.

2.4.2 PDNS - parallel/distributed ns

PDNS [25, 118] is an extended and enhanced version of the widely diffused network simulator ns [34], developed by the PADS research group at Georgia Tech. The simulator is based on a federated simulation approach and uses a conservative approach to synchronization. This choice allows to share the load between a set of

interconnected CPU but does not require to deeply modify the ns-code to support roll-backs. PDNS is currently supported and scalability tests has demonstrated a good scalability.

2.4.3 Scalable simulation framework (SSF)

The Scalable Simulation Framework (SSF) is developed as a common parallel simulation API suitable for but not exclusively for simulation of very large telecommunication systems [2]. The Dartmouth SSF (DaSSF) C++ based implementation relies on a process-oriented, conservative approach and is based on message passing (using MPI) to run on a combination of shared-memory and distributed-memory configurations. The next version of the tool named iSSF, will add support for HLA interoperability, real-time simulation, and human-interaction capabilities [12]. In [66] can be found interesting results in the simulation of wireless cellular networks.

2.4.4 Telecommunications Description Language (TED)

The Telecommunications Description Language (TED) [9] is an object-oriented language developed for modeling networks, it relies on the Georgia Tech Time Warp (GTW) which is a highly optimized optimistic parallel simulation kernel. TED has served as basis for the implementation of WiPPET, a parallel simulator designed to model the radio propagation, mobility and protocols of wireless networks [111]. Currently TED results no longer developed and supported.

2.4.5 SWiMNet

The SWiMNet is a simulation environment designed for large scale parallel simulation of wireless PCS networks [53]. It is based on a set of model components that can be combined to obtain the required PCS model. To exploit the maximum available architecture parallelism the real computation is based on precomputed events. The goal of the precomputed events is to explicit causality, taking advantage of this information in the following computation phase. This dual stage synchronization

mechanism is based on a conservative paradigm for the first phase (events precomputation) and optimistic (time warp) for the parallel simulation stage.

2.4.6 GloMoSim / QualNet

GloMosim [130] is a simulation environment for wireless and wired network systems based on the Parsec [39] parallel discrete-event simulation kernel. QualNet [27] is the commercial reincarnation of GloMoSim. QualNet is capable of monolithic simulations and parallel executions based on the conservative synchronization approach. It is reported as "scalable up to 10's of thousands of nodes" [27] and includes an extensive protocol model library. As an integration option is available an HLA & Threaded Communication Module.

2.4.7 JiST / SWANS

SWANS is a scalable wireless network simulator built atop of JiST, a high-performance discrete event simulation engine [13], both written in Java language and running on an unmodified Java Virtual Machine (JVM). SWANS claims to implement an efficient computation of signal propagation and the partitioning of node functionality into individual, fine-grained entities. These entities could be clustered across nodes to obtain a distributed simulation. The benchmarks reported on the JiST/SWANS claims for better results (reduced wall clock time) compared to GloMoSim and ns2.

2.4.8 OPNET

OPNET is a modeling and simulation platform of wired and wireless networks, it is object oriented and includes a very large library of implemented protocols. The tool supports discrete event, hybrid and analytical simulation, it claims to run in both sequential and parallel mode and to support HLA and co-simulation technologies [24].

2.5 High Level Architecture (HLA)

The High Level Architecture (HLA) is a general purpose architecture for simulation reuse and interoperability. The HLA was approved as an open standard through the Institute of Electrical and Electronic Engineers (IEEE) - IEEE Standard 1516 - in September 2000 [3, 10, 94].

One of the main forces behind the HLA development has been the requirement for simulation reusability and code reuse. For several years the USA Department of Defense (DoD) has faced with a decreasing budget, the military sector traditionally is a strong simulations user. In this situation the DoD has promoted the development of new technology able to increase the simulators modularity and interoperability. From an economic standpoint the simulation code reusability is valuable cost effective opportunity [71].

An HLA compliant simulation is composed by a set of federates. Each federate is a software component that interacts with other federates to form a simulation (federation). Previously built federates could be composed together to form new simulations. The interaction have to be performed through a distributed middleware named RTI. The common language of the federation is defined by a Federation Object Model (FOM). The Object Model Template (OMT) is used to define the structure of all FOMs. The HLA standard is composed by three parts: the rules, the OMT and the interface specification. Some rules have to be followed by the federates, others are about the whole federation. The HLA offers a set of services to the federates to build the federation and to perform the simulation: federation, declaration, object, ownership, time and data distribution management [10].

It is worth noting that the HLA defines an architecture and not an implementation, the interfaces are strictly defined but each RTI implementation is free to manage the execution details, define internal architecture, languages, technology and algorithms. Including both academic and commercial RTI implementation the set of available runtimes is not very crowded.

2.5.1 DMSO RTI

The Department of Defense-sponsored and developed Runtime Infrastructure (RTI) [4] was one of first and most diffused implementations but starting from September 30, 2002 it is no longer freely available and supported.

2.5.2 RTI NG Pro

The Virtual Technology Corporation [29] has continued the DMSO RTI development building a commercial, enhanced product with some new features and bugfixes.

2.5.3 Georgia Tech FDK

The Federated Simulations Development Kit (FDK) [6] is a software system that has been developed at Georgia Tech by the PADS research group led by Professor Richard Fujimoto. The software is formed by a set of specialized modules that can be composed to build a RTI with the requested set of features. The FDK communication module is optimized both for parallel and distributed simulation, the runtime would exploit shared memory if available and UDP/TCP for distributed environments.

2.5.4 MÄK RTI

The RTI built by MÄK Technologies [16] claims to be the "the fastest RTI available" [17]. An interesting aspect is the RTIspy, a way to inspect the RTI state variables at runtime. Recently the MÄK Technologies has released the MÄK Game-Link module, a software that allows to interoperate the famous gaming Unreal Engine with HLA or DIS simulations.

2.5.5 Pitch RTI

Traditionally the pRTI has been one of the first available commercial implementations. The RTI is written in Java and based on a partially centralized architecture.

In March 2003 it was certified as fully compliant with the IEEE 1516 standard [26].

2.5.6 XRTI

The Extensible Run-Time Infrastructure (XRTI) is a Java-based, Open Source implementation [5]. The XRTI current version is a prototype and implements a subset of the RTI interface, its target is to offer a testbed for improvements and extensions to the High Level Architecture. An interesting peculiarity are the autogenerated proxies: the simulation developer work could be made easier thanks to a set of autogenerated proxy classes. In this architecture the proxy is a software structure placed between the simulation model and the RTI.

2.5.7 OpenSkies Cybernet

The Cybernet is a communication architecture for multi-player games, gambling, stock market interactions, Internet-based classes, chat rooms and video teleconferencing. The technology behind the distributed architecture is based on the HLA standard and should allow great scalability and great reductions in the bandwidth load thanks to caching technology [23].

2.5.8 Chronos

Chronos is an advanced networking engine [14] based on standard HLA RTI interfaces and the DirectPlay technology [18]. The Chronos goals are wide but the runtime appears to be quite suitable for the development of Internet Games and it should be the first RTI providing a .NET programming interface compliant to the IEEE 1516 standard.

2.5.9 ERTI Mitsubishi Space Software Company

To the best of my knowledge no details are currently publicly available about this RTI implementation.

Chapter 3

The ARTÌS Middleware

"Well, it no use YOUR talking about waking him", said Tweedledum, "when you're only one of the things in his dream. You now very well you're not real."
"I AM real!" said Alice and began to cry.
Lewis Carroll, Through the looking-glass

This chapter illustrates the motivation, the preliminary design and implementation issues, of a new distributed simulation middleware named Advanced RTI System (ARTÌS) [35]. The aim of the ARTÌS middleware is to support parallel and distributed simulations of complex systems characterized by heterogeneous and distributed model components.

The ARTÌS design is oriented to support the model components heterogeneity, distribution and reuse, and to increase the simulation performances, scalability and speed-up, in parallel and distributed simulation scenarios. Another design issue of the ARTÌS framework is the dynamic adaptation of the interprocess communication layer to the heterogeneous communication support of different simulation scenarios. In this chapter it will be illustrated the guidelines and architecture that was considered in the design and implementation of the ARTÌS middleware.

The validation and the performance studies of the middleware are postponed to the following Chapter 4. Some case studies, and the distributed simulation of massively populated wireless ad hoc and sensor networks, will be presented.

3.1 Introduction

The design of complex systems composed by many heterogeneous components requires appropriate analysis methodologies and tools to test and to validate the system architectures, the integration and interoperability of components, and the overall system performances [30]. The performance evaluation of complex systems may rely on simulation techniques because the model complexity obtained with alternative analytical and numerical techniques often results in unpractical or unaffordable methods and computation [30, 39, 49, 75, 78, 130]. Well known sequential and monolithic event-based simulation tools have been created for analyzing general purpose system models (e.g. computation architectures, systems on chip, database systems) [32, 33] and more targeted system models (e.g. computer networks) [24, 34]. The problem with a sequential monolithic simulator is that it must rely on the assumption of being implemented on a single execution unit, whose resources may be limited, and it cannot exploit any degree of computation parallelism. To obtain a significant insight of a complex system, detailed and fine-grained simulation models must be designed, implemented and executed as a simulation process, often resulting in high computation and high memory allocation needs. This fact translates in computation and memory bottlenecks that may limit the complexity and the number of model components (i.e. the simulated system scalability) that can be supported by the simulation process. One solution to overcome these limitations can be found in parallel and distributed simulation techniques, in which many simulation processes can be distributed over multiple execution units. The simulation semantics, the event ordering and event causality can be maintained and guaranteed with different approaches (e.g. optimistic vs. conservative), by relying on distributed model-components' communication and synchronization services. Parallel and distributed simulation platforms and tools have been demonstrated to be effective in reducing the simulation execution time, i.e. in increasing the simulation speed-up. Moreover, parallel and distributed platforms could exploit wide and aggregate memory architectures realized by a set of autonomous and interconnected execution units, by

implementing the required communication and synchronization services. Examples of the Parallel and Distributed Discrete Event Simulation (PDES) approach can be found in [75, 78], e.g. Glomosim [130] based on PARSEC [39], Maisie [120], parallel and distributed implementations based on Network Simulator (ns-2) [34, 118] based on RTI-Kit [118], on ANSE/WARPED [114], Wippet [93], SWiMNET [54], and many others [100, 120]. More recently, the distributed simulation world agreed on the need for standards, and converged in the definition of a new standard, named IEEE 1516 [10, 60, 61].

Unfortunately, the need for distributed model-components communication and synchronization services may require massive interprocess communication to make the distributed simulation to evolve in correct way. Complex systems with detailed and fine-grained simulation models can be considered communication-intensive under the distributed simulation approach. As a result, interprocess communication may become the bottleneck of the distributed simulation paradigm, and solutions to reduce the cost of communication must be addressed by the research in this field [49, 75, 78, 128]. Additional research studies, aiming to exploit the maximum level of computation parallelism, dealt with dynamic balancing of logical processes' executions (both cpu-loads and virtual time-advancing speeds) by trading-off communication, synchronization and speed-up, both in optimistic and conservative approaches [65, 79, 124, 128]. The efficient implementation of interprocess communication is required as a primary background issue, to overcome the possible communication bottleneck of parallel and distributed simulations. The way interprocess communication can be sustained in distributed systems would depend mainly on the execution units' architectures and on the simulation system scenario. Recently proposed and implemented middleware solutions based on the IEEE 1516 Standard for distributed simulation and the High level Architecture (HLA) [10, 60, 61] have shown that the parallel and distributed simulation of massive and complex systems can suffer the distributed communication bottlenecks, due to suboptimal implementation of the interprocess communication services, over the simulation execution platform. In this chapter it will be proposed an overview of the design, preliminary implementa-

tion results and guidelines, for a new, parallel and distributed simulation middleware named Advanced RTI System (ARTÌS). The design of the ARTÌS middleware architecture is based on the guidelines provided by the analysis and evaluation of existing HLA-based RTI implementations, and on the observations about the sub-optimal design and management of distributed interprocess communication. Specifically, we oriented the ARTÌS design towards the adaptive evaluation of the communication bottlenecks and interchangeable support for multiple communication infrastructures, from shared memory to Internet-based communication services.

This chapter is organized as follows: in Section 3.2 are sketched the motivations for this study, and some comments on existing implementations; in Section 3.3 the design and architecture of the ARTÌS middleware will be described, with some emphasis on the implementation guidelines; Section 3.4 presents some conclusions and future work.

3.2 Motivation and preliminary discussion

In the following a short description of the motivations for this study will be given, and the comments that have been originated by the analysis of existing middleware implementations of the HLA-based distributed simulation middleware. Model components' reuse is considered a relevant issue to be supported in designing a new simulation system. On the other hand, model components design details may be confidential on behalf of the companies that designed them. The owner companies could be interested, under a commercial viewpoint, in allowing their models to be embedded as "black box" components for evaluating the integration analysis and compliance with other solutions. The open model-component source code could introduce the risk to reveal the confidential know-how in the component design solutions. A way to overcome this problem would be given by supporting the model component simulation in distributed way, and more specifically, over execution units local to the owner company domain. Distributed model components would simply export their interfaces and interactions (i.e. messages) with the simulation middle-

ware and runtime infrastructure (RTI) implementing a distributed simulation. This scenario would require that a general network communication infrastructure (e.g. the Internet) would support the message passing communication between distributed model components of a parallel or distributed simulation. This is the reason why we conceptualized a distributed simulation that could be performed over TCP/IP or Reliable-UDP/IP network protocol stacks, like in web-based simulations. Under the latter assumption, the distributed simulation platform is intended as a way to interconnect protected objects, instead of a way to improve the simulation speed-up. Other possible killer applications for such a distributed simulation middleware design would be the distributed Internet Gaming applications, gaining an even growing interest nowadays (see also Chapter 6). The opportune design of the simulation framework, based on the exploitation of the communication scenario heterogeneities and characteristics, could improve the overall simulation performance of distributed and remotely executed processes.

The most natural and efficient execution scenarios for parallel and distributed simulations often involve shared memory (SHM) and/or local area networks (LAN) as the infrastructures supporting inter-process communication and synchronization services. Nowadays, it is even more frequent the adoption of networked cluster of PCs, in the place of shared-memory or tightly-coupled multiprocessors, as the execution units of the distributed simulation, primarily for cost reasons. The aforementioned motivations for model reuse and wide distribution of the model component execution is demanding for a generalized support for inter-process communication, up to the Internet-based services. It is self-evident how the increase of the communication cost (i.e. the communication latency of local and wide area network-based communication) would result in a reduction of the simulation speed. In other words, any reduction of the communication-time cost would translate in more efficiency of the simulation processes. The communication-time reduction could play a fundamental role in determining the communication and synchronization overheads between the distributed model components.

As remarked in [49], a distributed simulation approach is not always guaranteed

to gain in performance with respect to a sequential simulation. The problem with the distributed simulation arises when a high degree of interactions is required in dynamic environments, mapping on distributed synchronization and inter-process communication. The basic solution to distribute the events information among interacting distributed components was the information flooding (broadcast) solution. This solution is quite immediate to implement over a generalized communication platform, but it was also originating the communication bottleneck effect for the distributed simulation. It was immediately clear that a reduction of communication would have been needed, by following two possible approaches: model aggregation and communication filtering. Model aggregation incarnates the idea to cluster interacting objects, by exploiting a degree of locality of communications that translates in a lower communication load than the one obtained in flat broadcast (that is, communication flooding) systems. Model aggregation can be performed by simplifying the model, or by maintaining the model detail. Solutions based on model simplification have been proposed, based on relaxation and overhead elimination, by dynamically introducing higher levels of abstraction and merging in system submodels [49, 114, 124]. These solutions allow a reduction of communication since the messages are filtered on the basis of the level of abstraction considered. Solutions preserving full model-detail have been proposed by dynamically filtering the event- and state-information dissemination. Examples can be found [49], based on interest management groups [124], responsibility domains, spheres of influence, multicast group allocation, data distribution management [31, 61], grid distribution and routing spaces [31, 61, 65], model and management partitioning [54]. These approaches rely on the reduction of communication obtained when the update of an event- or state-information (e.g. event and/or anti-message) does not need to be flooded to the whole system, but is simply propagated to all the causally-dependent components. This is the basis of publishing/subscribing mechanisms for sharing state-information and event-notifications between causally dependent components [31, 61, 118]. The solution provided in order to dynamically filter the communication among distributed objects was the ancestor of the Data Distribution Management

(DDM) concept realized and implemented in HLA-based solutions [10].

The High Level Architecture (HLA) is a middleware implementation based on standard (IEEE 1516) dealing with component-oriented distributed simulation [10]. The HLA defines rules and interfaces allowing for heterogeneous components' interoperability in distributed simulation. The definition of distributed model components (formally known as federates) with standard management APIs brings to a high degree of model re-usability. The HLA standard defines APIs for the communication and synchronization tasks among federates. The distributed simulation is supported by a runtime middleware (RTI). The RTI is mainly responsible for providing a general support for time management, distributed objects' interaction, attributes' ownership and many other optimistic and conservative event-management policies. The IEEE 1516 standard has gained a good popularity but still has not reached the planned diffusion. The main reasons, in my opinion, are the complex definitions and design work required to modelers. On the other hand, the preliminary implementations of distributed simulation middleware solutions and architectures were often too complex, too slow and required a great startup time to achieve the expected results. Specifically, since its definition, the IEEE 1516 Standard has been criticized about its structure and its effective ability to manage really complex and dynamic models [63]. By analyzing the existing RTI implementations, to the best of my knowledge, few currently available middleware solutions have been designed with some emphasis on the adaptive exploitation of the communication infrastructure heterogeneity. More specifically, the Georgia Tech RTI-kit [31] implementation has been realized by introducing some elasticity and optimization in the exploitation of the shared memory execution-system architecture, whereas many other implementations still rely on UDP or TCP socket-based interprocess communication even on a single execution unit. It is worth noting that rare implementations provided the source code to users, allowing them to configure the middleware on the basis of the user needs and execution-system architecture.

The support for heterogeneous communication services and architectures should be considered as a design principle in the implementation of a distributed simu-

lation middleware. Moreover, the adaptive optimization and management of the middleware communication layer realized over heterogeneous network architectures, technologies and services should be considered both in the initialization phase, and at runtime, in a distributed simulation process. Our ARTÌS implementation aims to be Open Source, and to provide an elastic, easy to configure adaptation of the communication layer to the execution system.

3.3 The ARTÌS middleware

The HLA implementation criticisms and the lack of efficient Open Source implementations are the main motivations behind the design and implementation of ARTÌS (Advanced RTI System). The main purpose of ARTÌS is the efficient support of complex simulations in a parallel or distributed environment.

The ARTÌS implementation follows a component-based design, that should result in a quite extendible middleware. The solutions proposed for time management and synchronization in distributed simulations have been widely analyzed and discussed. Currently, ARTÌS supports the conservative time management based on both the time-stepped approach, and the Chandy-Misra-Bryant algorithm. The optimistic time management (Time Warp) is implemented and currently under validation. The initial choice to support the conservative approach was a speculation on the highly unpredictable characteristics of our target models of interest [49], which would have led to frequent rollbacks. Anyway when our optimistic implementation will be completed, we plan to investigate this assumption, and compare the optimistic and conservative approaches as a future work.

In ARTÌS, design optimizations have been applied to adapt adequate protocols for synchronization and communication in Local Area Network (LAN) or Shared Memory (SHM) multiprocessor architectures. In my vision the communication and synchronization middleware should be adaptive and user-transparent about all optimizations required to improve performances. The presence of a shared memory for the communication among parallel or distributed Logical Processes (LPs) offers

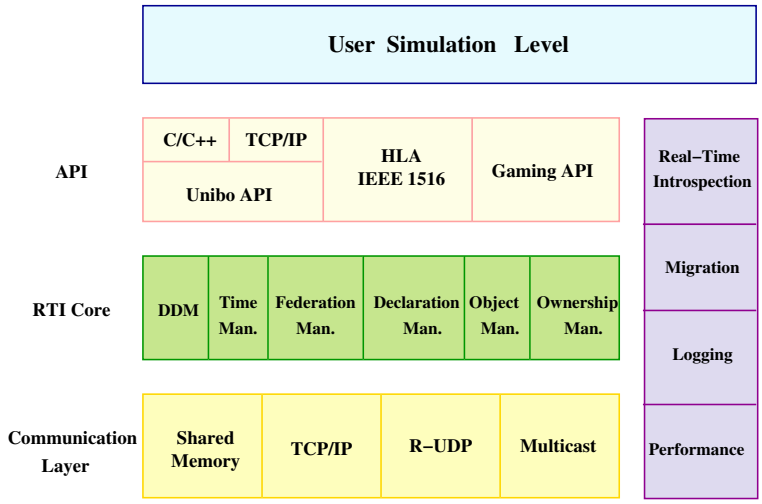

Figure 3.1: Logical architecture of the ARTÌS middleware

the advantage of low latency, and reliable communication mechanism. Interactions are modeled as read and write operations performed in shared memory, within the address space of logical processes. A memory access is faster than a network communication, but the shared memory itself is not sufficient to alleviate the distributed communication problem. To take advantage of the shared memory architecture, concurrent accesses to memory require strict synchronization and mutual exclusion, together with deadlock avoidance and distributed control.

The figure shows the structure of the ARTÌS middleware. ARTÌS is composed by a set of logical modules organized in a stack-based architecture. The communication layer is located at the bottom of the middleware architecture, and it is composed by a set of different communication modules. The ARTÌS middleware is able of adaptively select the best interaction module with respect to the dynamic allocation of Logical Processes (LPs) in the execution environment. The current scheme adopts an incremental straightforward policy: given a set of LPs on the same physical host, such processes always communicate and synchronize via shared memory. To implement these services we have designed, implemented and tested many different solutions. The first implementation was based on Inter Process Communication (IPC) semaphores and locks. This solution was immediately rejected both for per-

formance reasons (semaphore and locks introduce not negligible latency), and for scalability reasons (since the number of semaphores that could be instantiated in a system is limited and statically controlled in the operating system kernel). Among other possible solutions (e.g. we also considered busy-waiting) the current ARTÌS synchronization module works with "wait on signals" and a limited set of temporized spin-locks. This solution has demonstrated very low latency and limited CPU overhead, and it is really noteworthy for good performances obtained in multi-CPU systems, good scalability and also because it doesn't require any reconfiguration at the operating system kernel level. In ARTÌS, two or more LPs located on different hosts (i.e. no shared memory available), on the same local area network segment, communicate by using a light Reliable-UDP (R-UDP) transport protocol over the IP protocol. Ongoing activity is evaluating the use of raw sockets for R-UDP data segments directly encapsulated in MAC Ethernet frames (i.e. bypassing the IP layer). The drawback of this solution is that it could be adopted only within a common LAN segment technology. Two or more LPs located on Internet hosts rely on standard TCP/IP connections. Ongoing activity at this level is performed by considering the exploitation of reliable multicast-IP solutions. The ARTÌS runtime (RTI) core is on the top of the communication layer. It is composed by a set of management modules, whose structure and roles have been inherited by a typical HLA-based simulation middleware, compliant with the IEEE 1516 Standard. The modules currently being under the implementation phase are: the Data Distribution Management (DDM) in charge of managing the dynamic subscription/distribution of event and data update messages, the Time Management module, the Federation Management, Declaration Management, Object Management and Ownership Management modules in charge of administrating all the remaining management issues accordingly with the standard rules and APIs. The ARTÌS runtime core is bound to the user simulation layer by modular sets of application programming interfaces (APIs). Each API group was included in order to allow a full integration and compliance of many distributed model components with the ARTÌS middleware. The Standard API is implemented as the HLA IEEE 1516 interface: this will allow the

integration of IEEE 1516 compliant models to the ARTÌS framework. Only a subset of the full Standard API is currently implemented in ARTÌS. The new set of APIs of ARTÌS is called the *University of Bologna APIs* (Unibo APIs). These APIs are currently designed and implemented to offer a simpler access to distributed simulation services than the Standard APIs. This would make easier and simpler for the modelers to create and instantiate a distributed simulation with ARTÌS, than with the Standard APIs. We planned also to include in ARTÌS an API set specific for Internet Gaming applications, whose design is still in preliminary phase.

Additional orthogonal modules are planned to be dedicated to other specific features, oriented to the adaptive runtime management of synchronization and communication overheads. As an example, a real-time introspection mechanism would be devoted to offer an internal representation of the middleware state while the simulation is running. Logging and Performance modules would support the user simulation with online traces, statistics and runtime data analysis. Previous research works shown that more efficient simulation of dynamic models can be obtained by introducing additional software components implementing distributed model entities migration [49]. The Migration module is orthogonal in the middleware, with the target to reduce runtime communication overheads accordingly with the coordination supported with other peer modules. To this end, is included in ARTÌS a dynamic model-entity migration support, inspired to the prototype framework built for the HLA-based middleware on [49]. By defining a dynamic allocation of simulated model entities over different physical execution units can be obtained speed-up improvements (and communication overheads reduction) that could be further optimized in ARTÌS, when supported by opportune coordination of the migration modules.

3.4 Conclusions and future work

This chapter has illustrated the motivation, the preliminary design and implementation issues of a new parallel and distributed simulation runtime. The goal to build an efficient runtime, to support a wide set of APIs and to achieve IEEE standard com-

patibility is quite ambitious. The runtime implementation is far from completeness but the middleware is sufficiently complete to support the simulation of massively populated wireless networks (see Chapter 4) and to work as testbed for our research on speed-up improvements.

In the future we plan to complete our implementation and to optimize the current software. A wide set of features has been externally developed and we are going to integrate them in the middleware. Our current efforts are toward the research of better algorithms for the management of the distributed event queues, further reducing the communication overhead and the support of High Performance Computing (HPC) architectures with hundreds of CPUs and non uniform shared memory [1]. Actually we are working to raise the status of our software from "lab development" to a "public release" tool, improving usability and documentation.

Chapter 4

Simulation of Large Scale Wireless Systems

"... [I] heard a voice of many angels... the number of them was ten thousands of ten thousands and thousands of thousands..."
The Book of Revelation

The simulation of ad hoc and sensor networks often requires a large amount of computation, memory and time to obtain significant results. The parallel and distributed simulation approach can be a valuable solution to reduce the computation time, and to support model components' modularity and reuse. In this chapter it will be presented a testbed evaluation of the ARTÌS middleware for the simulation of large scale wireless systems. To realize a testbed evaluation of the considered framework a set of wireless systems' models were implemented and investigated. Specifically, two classes of widely investigated wireless models were identified: mobile ad hoc, and static sensor networks. In this chapter are presented the performances of the simulation framework, with respect to the heterogeneous set of execution architectures, and the modeled systems' characteristics. Results demonstrate that the framework leads to increased model scalability and speed-up, by transparently adapting and managing at runtime the communication and synchronization overheads, and the load balancing.

4.1 Introduction

The research in the field of wireless systems, protocols and architectures has been characterized by the need to investigate even more complex and detailed models of wide, scalable and integrated systems. Many standards and system architectures for wireless systems have been proposed, and are currently being deployed and developed, while other solutions are currently under the design and analysis phases for future deployment. Wireless networks‘ architectures and wireless sensor systems are currently under analysis to obtain insights and guidelines governing many relevant design issues: as an example, system architecture and management choices, protocols‘ design, dynamic self-configuration and adaptation to system dynamics, systems and protocols‘ interoperability and co-existence, system scalability and system fault-tolerance and lifetime. To obtain valuable insights of the investigated indices, researchers would require intuitive, accurate and fine-grained methodologies and tools for the analysis. Because of mathematical intractability and the model complexity, simulation-based investigation of wireless systems is often preferred to numerical and analytical resolution methods [116, 117, 120]. Simulation makes more practical the modeling and investigation of complex and dynamic scenarios, often characterized by multiple correlated factors, “memory effects” of the system states, and dynamic causal effects. Under such conditions, a simulation model would allow a level of detail that exceeds the detail level that could be obtained in most tractable mathematical models. A modular simulation modeling makes it possible the model components‘ reuse and composition, and works in favor of a correct system design, together with the possibility of a preliminary functional-test and interoperability analysis that would result in a fast system deployment.

On the other hand, two problems appear to limit the adoption of simulation techniques for the analysis of complex systems: i) the limitation of affordable-cost simulation execution architectures (mainly memory and computational power) [116, 117, 120], and ii) the scarce possibility of model components‘ reuse and model composition among heterogeneous simulation models and tools. As a consequence,

the research for tools and new methodologies for standard- and module-based modeling and simulation of large-scale and complex wireless networks has received a great attention by the research community, and has led to some interesting results [39, 42, 52, 54, 79, 84, 92, 93, 100, 113, 115, 124]. Currently, it is widely recognized that many of the most adopted tools for simulation of wireless systems (e.g. Network Simulator, ns2[34]) suffer the memory limitation of the execution architecture (which translates in a limitation of the model complexity and scalability). Also, they suffer the great computation time required to complete the simulation processes [107]. The fact that wireless networks models may be complex and may include a potentially huge number of simulated and dynamically interacting components would represent an amplifier of the modeling limitations due to memory constraints of current tools and simulation architectures. Specifically, under the computation viewpoint, the simulation of wireless systems‘ models may require a long time, due to the execution of complex behaviors and state updates required on behalf of many model components. Many different model factors may introduce additional computation requirements for implementing detailed model components behaviors: e.g. mobility patterns and topology changes, layered communication protocols, resource sharing, interference effects, among others. As a result, large scale and complex simulation models are often unpractical to simulate on a single-processor execution unit, because of huge memory requirements and large amount of time required to complete the simulation runs [116, 117]. Parallel and distributed models and architectures may be a viable alternative to reduce memory bottlenecks through distributed memory hierarchies, and to obtain simulation speed-up thanks to the parallel execution of computation tasks [34, 39, 53, 54, 75, 78, 92, 93, 100, 118, 120, 130].

As seen in the previous chapters the High Level Architecture (HLA) is a standard (IEEE 1516) for distributed simulation. Most of the RTI HLA implementations available so far have gained a good interest and diffusion, but have also been subject to some criticisms. The investigation of new features and services, and the lack of Open Source runtime implementations are the main motivations behind the design and implementation of the ARTIS (Advanced RTI System) middleware, which was

illustrated in the previous Chapter 3.

The main bottleneck arising in a distributed simulation framework is given by the communication overheads to realize the event-message distribution and synchronization services between a set of distributed model entities. The communication overhead due to the message passing required for the parallel and distributed simulation could nullify all the performance gain obtained by parallel executions. The ARTÌS framework is designed to realize the dynamic adaptation of the interprocess communication layer to the heterogeneous communication support offered by possible different simulation-execution architectures. Specifically, we oriented the ARTÌS design towards the adaptive evaluation of the communication bottlenecks and support for multiple communication infrastructures and services, from shared memory to Internet-based communication. ARTÌS has been also integrated in a framework with another middleware, named Generic Adaptive Interaction Architecture (GAIA). Basically, GAIA implements a simple model components‘ migration mechanism, preliminary proposed on the top of HLA-based distributed simulations [49]. The HLA standard and existing RTIs do not define component migration facilities, and preliminary research activity was made on this topic [101, 106]. GAIA includes a heuristic migration policy, whose aim is to dynamically partition and allocate the interacting model components over many Logical Processes (LPs), respectively executed over a set of multiple, distributed execution units. The composition of ARTÌS and GAIA realizes a framework for parallel and distributed simulation, characterized by adaptive reactions to dynamic systems‘ behavior, and oriented to the communication-overhead reduction. In this chapter, the prototype implementation of the ARTÌS and GAIA framework is outlined. A set of testbed-evaluation results are presented to express the potential of ARTÌS with and without GAIA over heterogeneous execution architectures, and for two classes of relevant wireless systems‘ models: wireless mobile ad hoc networks and wireless sensor networks.

The chapter structure is the following: in Section 4.2 will be outlined the guidelines and motivations for the modeling and implementation of distributed simulation of wireless systems. In Section 4.4 the GAIA framework is introduced. In Section

4.5, two wireless system's models are described. In Section 4.6 will be presented a set of simulation results. In Section 4.9 conclusions will be summarized.

4.2 Distributed simulation of wireless systems

Wireless systems are highly dynamic systems where the interactions are subject to fast changes driven by the system evolution. Given a wireless system model, it is quite natural for modelers to implement and recycle a set of model components, each one realized by a composition of software modules, to obtain the global system model. Every autonomous model component (e.g. a wireless node or sensor) is then required to mimic the interactions (i.e. the causal effects of events) with all the neighbor components in the modeled scenario. Two inherent characteristics of wireless systems play an important role in the modeling and simulation viewpoint: i) the hosts‘ mobility and ii) the open broadcast nature of the wireless transmissions.

As an example, topology changes due to simulated hosts‘ mobility map on causality effects in the “areas of influence” of each mobile device. This may result in dynamically shaped causality-domains, which map on the interaction-scheme of the distributed model components. Intuitively, given two or more neighbor-hosts sharing the wireless medium, the causal effect of a signal-interference event due to the “open broadcast” nature of the wireless transmissions could result in a chain of local-state events up to the high protocols‘ layers [125]. In our modeling approach, we define a model entity as the data structure defined to model a Simulated Mobile Host (SMH).

A high degree of causality in the simulation of the wireless hosts‘ communication is driven by the local-topology interaction (i.e. transmissions) between neighbor hosts [11, 125]. Under the modeling and simulation viewpoint, if a SMH changes its position, it will eventually interact with a new community of neighbor hosts. A certain degree of time-locality among neighbors‘ communications can be considered an acceptable assumption in many wireless system models, depending on the communication load and the mobility model assumptions. The system and model dynamics

can be influenced by motion model and speed, and also by the SMHs density.

To realize a correct evolution in a parallel or distributed simulation, under the event-causality viewpoint, every model components‘ interaction should be notified as an event-message to all the causally dependent model components. In a distributed simulation, this task is usually managed by a runtime event-message distribution mechanism. Complex systems with detailed and fine-grained simulation models can be considered communication-intensive under the distributed simulation approach. As a result, interprocess communication may become the bottleneck of the distributed simulation paradigm. The way interprocess communication can be supported in distributed systems would mainly depend on the execution units and on the simulation system resources, architectures and characteristics. As an example, message passing communication can be performed efficiently over shared memory architectures, while it would require medium/high communication latencies over local and wide area network communication services.

The physical clustering of interacting model components on a shared memory architecture could result in the advantage to exploit the most efficient message passing implementation. Unfortunately, in wireless mobile networks any optimal, static clustering and allocation of model entities, based on the current component-interaction scheme, will become immediately suboptimal, due to the dynamics of the model interactions (e.g. SMH mobility). The approach used in currently available implementations of parallel and distributed simulation frameworks is to passively detect the model component interactions, by adapting the event message distribution accordingly. No background optimization is based on the heterogeneous characteristics of any available communication infrastructure. This observation was a motivation for the design of our simulation framework.

The event-message distribution of a distributed simulation requires a dynamic definition of publishing/subscribing lists, or the implementation of a complete state-sharing information system. On the other hand, a dynamic approach for the event-distribution and state-information-updates (e.g. dynamic lists and subscription groups) would lead to additional communication and management overheads. In

some scenarios, the communication cost of list-updates or fine-grained events‘ communication between a dynamically variable set of components, could make attractive a complementary approach. As an example, when the system communication infrastructure is characterized by significant performance asymmetry (e.g. shared memory vs. LAN communication), like in networked clusters of PCs, the migration cost needed to dynamically cluster the set of interacting components over a single Physical Execution Unit (PEU) could become attractive. his would be even more attractive if the following three assumptions could be satisfied: i) components‘ migration could be implemented incrementally as a simple data-structure (i.e. state) transfer, ii) the component state would be comparable with the amount of data exchanged for interactions, and iii) the object‘s interaction scheme would be maintained for a significant time (i.e. time-locality).

4.3 The simulation execution testbed

The simulation testbed consists of a distributed, discrete-event simulation of model components. Model components are executed as logical processes over a set of physical execution units (PEUs), connected by a physical LAN network.

The execution architecture for our experiments is realized by 3 PEUs each one equipped by Dual Xeon Pentium IV 2800 Mhz, with 3 GB RAM, and one PEU equipped by Quad Xeon Pentium IV 1500 Mhz, with 2 GB RAM, all connected by a Fast Ethernet (100 Mb/s) LAN, and all equipped with Debian GNU/Linux OS with kernel version 2.6.

The design approach is mainly focused on the adaptive communication-reduction between the PEUs where Logical Processes (LP) are executed. Every LP is statically allocated and executed on a single PEU. Specifically, one single LP cannot be split over two or more PEUs, more LPs can be executed over a single PEU, and LPs cannot be migrated between PEUs.

Every LP is managed by a runtime simulation core (ARTÌS) as a single simulation component. On the other hand, a single LP is implicitly formed by a set

of threads, each one managing and updating the state (i.e. local data structures) of a set of Simulated Mobile Hosts (SMHs). A communication between wireless hosts can be modeled as a set of interactions (i.e. message-events) between any couple of adjacent SMHs. Since a wireless communication must be always modeled as a broadcast within a limited local transmission range, this requires that each SMH within a variable range would be notified with the transmission-related event-messages. Each event would result in a multiple set of one-to-one interactions (i.e. event messages) among local SMHs. If the sender SMH and its neighbors belong to the same LP (i.e. they are executed on the same PEU), or if they belong to different LPs implemented over the same PEU, then their interactions can be considered local (e.g. shared memory communication) and do not involve any physical network communication. On the other hand, every interaction involving participants implemented over foreign LPs (e.g. LPs implemented over different PEUs) may require time-expensive physical network communication. By reducing the physical network communication we can reduce the synchronization delays. By clustering neighbor SMHs within the same LP, or within the LPs executed over the same PEU, we close the causal interactions and system communication within the PEU where the interacting LPs (and their respective SMHs) are executed. In addition, clustered interacting SMHs would limit interactions with the management layers of the ARTÍS middleware, by further reducing the computation and communication overheads. To sum up, by limiting the network communication in favor of the local (shared memory) communication, the wall clock time required by the simulation runtime to achieve full synchronization would be reduced. This would make it possible to obtain a simulation speed-up.

A static approach could be adopted to optimally distribute the SMHs within the LPs in the simulation initialization phase. The optimal solution for allocation is hard to find and could be defined in many ways, depending on the targeted overheads' reduction. Typically, the optimality is defined with respect to latency (to reduce the physical network communication cost) or computation (to obtain an optimally balanced execution parallelism). Anyway, this should be explicitly performed offline

by the modeler, on the basis of the modeling assumptions. Moreover, as it will be demonstrated in the final results, the model dynamics (e.g. the SMH mobility) would make the initially optimal distribution less effective after few simulation steps. This result may translate in a performance degradation for the simulation speed-up, mainly due to the increasing cost of communication and synchronization required between distributed model components (logical processes). In our approach the optimization is dynamically performed at runtime, by the proposed simulation middleware, by migrating the SMHs between LPs. In this way, the modeler is relieved by the optimization task, and the system converges towards a balanced, tuneable and pseudo-optimal model components‘ distribution driven by the model interaction scheme. If a time-locality is assumed in the interaction between neighbor hosts, it could be convenient to migrate the foreign SMH to the LP (and to the PEU) where its new neighbors are located, by reducing in this way the cost of successive interactions. This assumption is typically verified in MANETs, e.g. most routing protocols are based on a “proximity” concept to decide the routing path of communications, and such communications usually last for a significant time, following a bidirectional session-based scheme. The effect of the time-locality of the causality effect inside each logical process will be investigated in the final section of this chapter, by varying the SMH mobility speed.

4.4 The distributed simulation framework

4.4.1 The advanced RTI system (ARTIS)

The main purpose of ARTÌS is the efficient support of complex simulations in a parallel and distributed environment. The ARTÌS implementation and design was described in Chapter 3

4.4.2 The generic adaptive interaction architecture (GAIA)

The PDES simulator built to obtain an experimental testbed of our proposal is based on a distributed architecture made by a set of logical processes glued together by the ARTÌS middleware. In our preliminary design we adopted the High Level Architecture (HLA) DMSO (Department of Military Simulation Office, US Department of Defense) implementation RTI-1.3NGv3.2 as the basis for our work. On top of the HLA RTI we built a middleware extension called Generic Adaptive Interaction Architecture (GAIA). GAIA provides the interaction to the simulation core, the location and distribution data management, the random number generator, tracefile-logging and other simulation facilities.

The target of GAIA is to provide migration and service APIs to the simulation developer. Because of the unavailability of DMSO RTI source-code, the GAIA facilities were initially provided as an external middleware on top of the DMSO RTI. The development of ARTÌS middleware has permitted to merge the GAIA framework within the runtime core, still reducing the runtime execution overheads.

The SMH models are implemented as code with data structures to define and maintain the SMH state information. GAIA migrates the "data structure", i.e. the state information of SMHs between LPs. This required to design and to implement a migration layer for the "state" of the SMH model entities between LPs. The ARTÌS runtime has been extended to execute static models and to exploit migration by means of a small set of Application Programming Interfaces (APIs) providing migration services for migration-enabled models. To test our framework we implemented a time-stepped, conservative, parallel and distributed discrete-event simulation of two classes of models for mobile wireless systems. By following the guidelines obtained in, the ARTÌS runtime has been designed and implemented as an alternative to HLA runtime, and the GAIA middleware has been completely reimplemented, with both the migration and the load-balancing heuristics completely redesigned. Moreover, the composition of GAIA with ARTÌS results in lower management overheads and greater speed-up than the the ARTÌS framework without the migration support.

4.5 Wireless systems' model definition

In the following it will be described two classes of wireless systems' models that was considered in the testbed evaluation of the ARTÌS and GAIA framework. The two classes of models have been selected i) because they represent two examples of widely studied systems and ii) because they capture most of the complementary characteristics of wireless simulation systems about mobility, communication load, management overheads, resource limitations. This will let us to obtain results about the optimization and speed-up achieved by our simulation framework, based on the exploitation and adaptation to many variable model characteristics.

4.5.1 The mobile Ad Hoc network's model

The definition of a mobile ad hoc network model is basically devoted to study the effect of hosts‘ mobility and high communication loads assumptions under the modeling viewpoint. It was assumed a highly scalable number of simulated mobile hosts (SMHs), each one following a Random Mobility Motion model (RMM). This motion model is synthetic and far from reality, but the choice was driven by the unpredictable and uncorrelated mobility pattern of SMHs. This is the worst case analysis for the presented mechanism, because any heuristic definition cannot rely on any assumption about the motion correlation and predictability of SMHs. The only correlation effect exploited in our mechanism is given by the “time-locality” of communication sessions between neighbor-hosts. Given the framework definition, my feeling is that any other widely used motion model, like any restricted, correlated or Group Mobility Model, would result in better results than the adopted RMM model, for any migration heuristic. The RMM model is defined by SMHs swinging between mobile and static epochs. At the beginning of each epoch, every SMH decides to stay or to change its mobile or static status, by following a geometric distribution with parameter p=1/2. When entering a mobile state, new, uncorrelated and uniformly-distributed direction and speed are randomly selected and maintained up to a static epoch. The cycle is repeated for the whole simu-

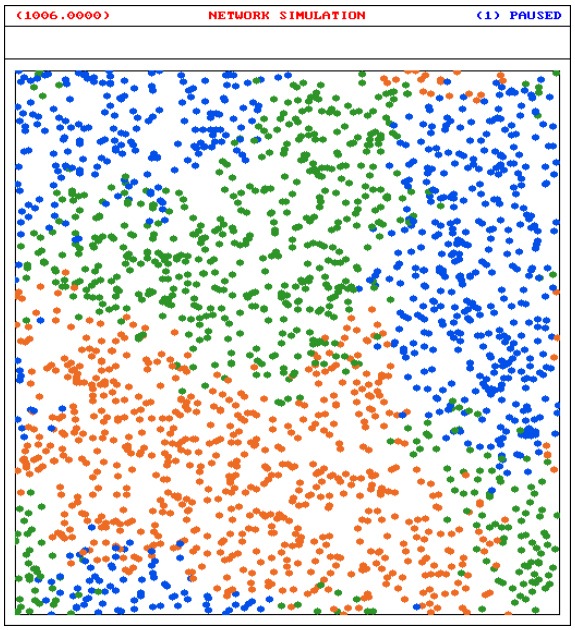

Figure 4.1: A snapshop of a Mobile Ad Hoc network with 1000 SMHs dynamically allocated by GAIA over 3 PEU. Dot colors define the PEU where the SMH is executed.

lation run by every SMH. Sometimes we considered motion sub-models related to the motion speed, i.e. high speed (25 spaceunits/timestep), and lower speed (10 spaceunits/timestep). To stress the migration scheme, it was also used an extreme sub-model with very high speed (100 spaceunits/timestep).

Space is modeled as a torus-shaped 2-D grid-topology, 10.000x10.000 spaceunits, populated by a constant number of mobile SMHs. SMHs are randomly and uniformly distributed in the simulated area (see dots positions in Figure 4.1). The torus space topology, indeed unrealistic, is commonly used by modelers to prevent non-uniform SMHs‘ concentration in any area. This allows to evaluate the mechanism behavior in a worst case scenario, where the clustering of SMHs is not trivially determined by high concentration in small areas. These are stressing examples for our mechanisms, because they will lead to a high migration overhead, given the motion model defined. The simulated space is wide and open, without obstacles. The modeled communication between SMHs is a constant flow of ping messages

(i.e. constant bit rate), transmitted by every SMH to all neighbors within a wireless communication range of 250 spaceunits. Again, this choice is stressing the migration mechanism under the mobility effects of continuously transmitting SMHs. In the defined scenario, since the SMH migration policy is evaluated on the basis of the local and remote interaction (i.e. communication), no communication translates in no migration needs, hence no additional communication, synchronization and migration overheads. The rate of ping messages is constant because it is the control parameter for communication: increasing/reducing the ping rate would be equivalent to change the interaction rate. For the future, and extended version of the model is planned, with the real implementation of message flows, routing protocols and applications as a future work. Currently, the host model implements the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) Medium Access Control protocol of the IEEE 802.11 Distributed Coordination Function. Anyway, this model is not used in this investigation because we are only interested in modeling the basic interaction, at the physical layer, which is given by the event of a channel occupancy due to local ping messages among neighbor hosts. It is worth noting that additional local computation would be required by adopting more detailed and complete protocol stacks implementations over the SMH model entities, resulting in additional advantages of parallel execution.

4.5.2 The sensor network model

The second model considered in this testbed evaluation is based on a wireless sensor network system. In this model we are interested to test the model scalability, by showing results in a system with up to 40.000 sensors. The most important feature of this system with respect to the mobile ad hoc network, is given by the fact that sensors are static. They are randomly placed with uniform distribution in the simulated area (see Figures 4.2 and 4.3).

For maintaining the protocol behavior and average connectivity, the area size is variable such that the sensor density is constant in all experiments (approximatively 1 sensor in 10x10 space units). The communication range of each sensor is 15

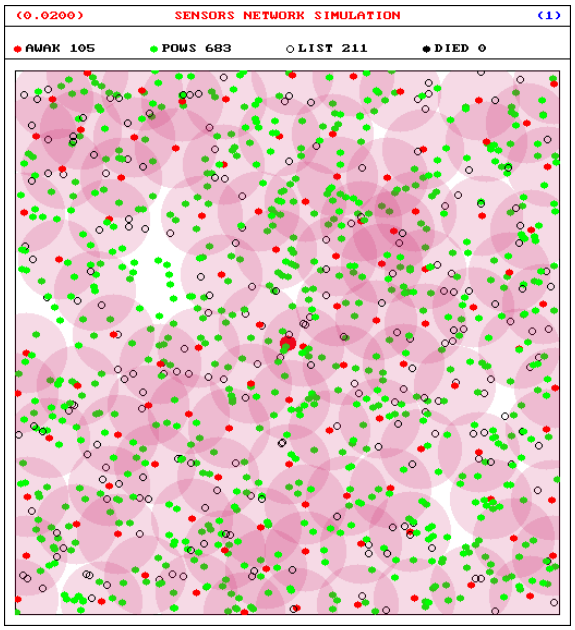

Figure 4.2: A snapshot of a wireless sensor network with 1000 sensors. Dot colors refer to sensor state: red=active, green=power saving, white= listening.

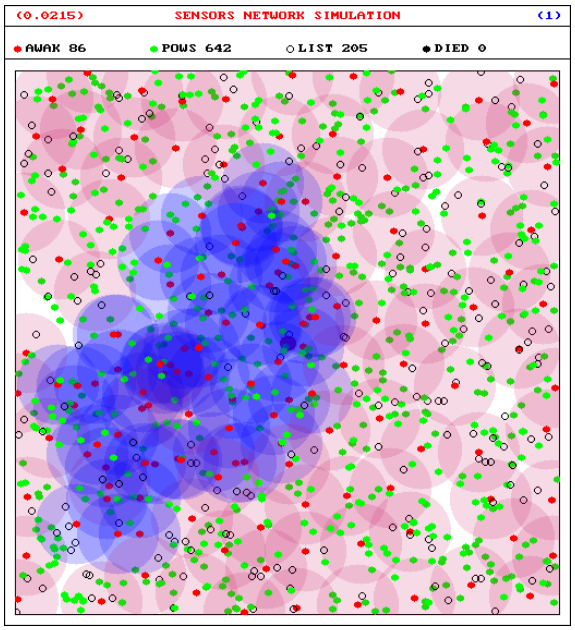

Figure 4.3: A snapshot of a wireless sensor network with 1000 sensors while propagating an alert message.

space units. Every sensor implements a "pressure variation" detector and sends broadcast alerts that are flooding towards a set of target detection points. To maximize the network lifetime, every sensor implements a power saving mechanism that adaptively manages the sensor state. A sensor can be active, listening and in power saving state. A new Medium Access Control (MAC) scheme, whose definition is out of the scope of this thesis is currently investigated by adopting the defined model. Under the modeling and simulation viewpoint this model is complete and provides detailed information about the system behavior, both for sensor network management, communication, resources' utilization and network lifetime indices.

4.6 Experimental results

In this section will be presented the results of some testbed simulation experiments executed to test the ARTÌS distributed simulation framework, and the GAIA middleware. The experiments have been performed over heterogeneous execution infrastructures and scenarios, and have involved two different classes of wireless systems models. The motivation for this study is given by the evaluation of the adaptive self-configuration of the ARTÌS framework and the GAIA middleware, executed in transparent way with respect to the model and execution infrastructure characteristics. The target indices to be evaluated include: the observation of migration overheads related to model dynamics under GAIA, the advantages obtained by GAIA and ARTÌS under the adaptive communication overheads reduction, and the speed-up obtained by our framework under variable execution scenarios and under variable modeling assumptions for the simulated wireless systems.

The execution architecture for the experiments was previously described. Multiple runs of each experiment was performed, and the confidence intervals obtained with a 95% confidence level are lower than 5% the average value of the performance indices shown.

In the following it is defined as M the number of physical execution units (PEUs) supporting the simulation execution, and as N the total number of logical processes

(LPs) implemented. With "migration ON" or "migration OFF" is identified a distributed simulation with the GAIA migration heuristic turned ON and OFF, respectively. All the performed experiments were started with a pseudo-random, uniform distribution of a variable number of SMHs for both the ad hoc, and the sensor networks models. Initially, the set of SMHs is randomly allocated over the set of PEUs, without any optimal allocation. The choice of the initial random distribution allows to analyze the transient dynamic effect of the GAIA migration mechanism. In [49] it was shown that the random distribution would be asymptotically obtained if migration is disabled, starting from any initial (and optimal) allocation scheme, due to the SMHs' mobility. Most of the figures presented show transient behavior of the performance indices, because this describes the dynamics and fast convergence effect of the proposed mechanisms. Steady-state results have been also discussed to define the asymptotical behavior of the proposed framework.

4.7 Mobile ad hoc network's simulation

Figure 4.4 shows the transient number of model components (SMH) migrations performed by the GAIA middleware during the initial phase of a distributed simulation of the mobile ad hoc network model. The model is composed by 5000 SMHs, randomly distributed in the simulated area, and randomly allocated over three PEUs. The migration heuristic of GAIA begins to migrate the SMH model components between PEUs after a warmup (observation) phase of 50 timesteps.

The Figure 4.4 shows the transient number of migrations performed between the three PEUs in every timestep, based on the average speed value of SMHs in the simulated mobile ad hoc network (i.e. 10, 25 and 100 m/s). The SMH speed here is just a modeling factor to stress the simulation and it is not expected to be realistic. It can be observed that in the initial phase the GAIA middleware induces a peak of model components reallocation aiming to cluster the interacting SMHs over the same PEU. The resulting model component allocation over PEUs at timestep 1000 would be similar to the distribution shown in Figure 4.1. In Figure 4.1, one dot represents

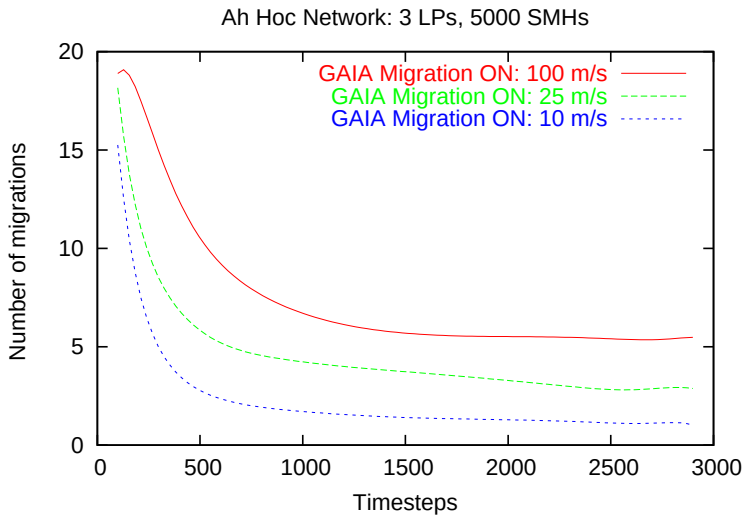

Figure 4.4: Transient number of GAIA migrations per timestep, with respect to modeled mobility parameters.

a SMH in the simulated area, and the color of dots indicates the PEU where the SMH is executed. It is clear the clustering effect obtained by GAIA migrations after the initial transient phase. At the steady state, the SMH (dots) mobility would smoothly require a continuous adaptation and migration of SMHs moving out of the context of SMHs executed over the local PEUs. Figure 4.4 indicates that the higher the mobility (speed) of SMHs, the higher is the reallocation rate required by GAIA to optimize the degree of local communications within the PEUs.

Figure 4.5 shows the Local Communication Ratio (LCR) of messages originated by the simulation in the same scenario considered in Figure 4.1 and 4.4. The LCR is intuitively defined as the percentage of message passing required by the simulation execution which is local to each one of the three PEUs adopted for the execution. Given the ARTÌS design and assumptions, local message passing translates in efficient shared memory communication within each PEU, as an alternative to less efficient and time consuming network communications. Figure 4.5 shows that the GAIA migration allows to obtain a steady-state percentage of local communications around 85% in the considered scenario. Figure 4.5 also indicates that the mobility of SMHs have less or no effect on the LCR index when GAIA migration is active. This

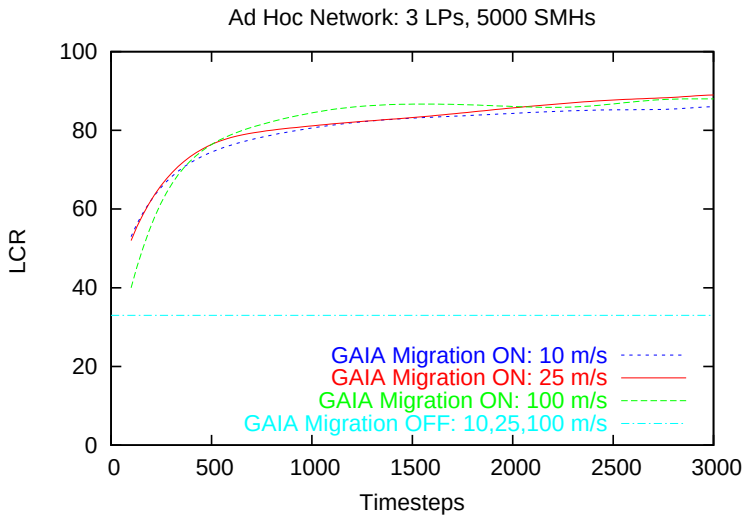

Figure 4.5: Transient percentage of local communications per timestep, with respect to modeled mobility parameters, with and without GAIA migration.

is due to the adaptive effect of GAIA migrations at runtime, shown in Figure 4.4. The same simulation scenarios with GAIA migration OFF result in a LCR index which is around 33%, as expected when interacting SMHs are randomly allocated over three PEUs.

Figure 4.6 shows results about the speed-up of the distributed simulation under the mobile ad hoc modeling scenario considered above. The speed-up is shown as a transient index, by averaging the consecutive speed-up indices calculated over separated and adjacent simulated time windows. This allows to evaluate the transient effect of the speed-up in the initial phase, and when GAIA is switched OFF at runtime. The monolithic scenario is considered as the normalization value for speed-up evaluation. A monolithic simulation is intended as a single logical process (LP) executed over a single PEU. In our implementation, when analyzing the simulator performances, it is considered the monolithic execution platform as equivalent to a single sequential simulator. This assumption is not completely true in practice, because a really small biasing effect is introduced by the ARTÌS middleware in background, anyway the biasing is really low, and this assumption allows us to study comparable systems and models. When the simulation is executed over 3

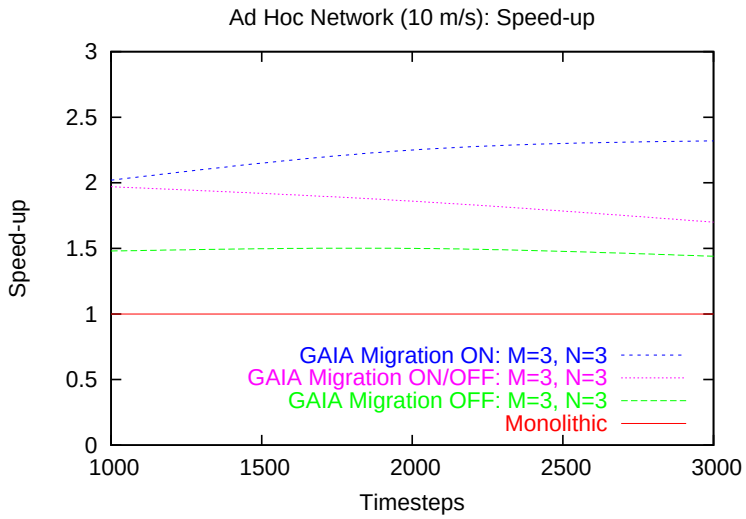

Figure 4.6: Transient speed-up effect over ARTÌS with and without the GAIA migration mechanism. Average SMH speed: 10 m/s.

PEUs (M = 3) and each PEU implements a single LP (N=3) with GAIA migration OFF, the speed-up obtained is around the value 1.5 with respect to the monolithic execution scenario. In the same scenario with GAIA migration ON, the speed-up starts around the value 2 and it increases up to 2.3 by the effect of GAIA dynamic reallocation and the increase of local communications. The curve labeled "GAIA Migration ON/OFF" shows the effect of degradation of the speed-up obtained when, after the initial reallocation of GAIA, the GAIA migration is switched off: it is clear how the dynamic effect of SMH mobility (whose average speed is 10 m/s) realizes a transient mutation of interactions (i.e. communication) from local to external for the PEUs, by decreasing the speed-up.

Figure 4.7 shows the same indices of Figure 4.6, with the only difference given by the average speed of the modeled SMHs: from 10 m/s in Figure 4.6 to 25 m/s in Figure 4.7. As expected, the high modeled speed translates in less "time-locality" effect of local interactions. This reduces a little the speed-up index obtained, because GAIA introduces less local communication advantages. On the other hand, the relative differences among the considered scenarios and mechanisms remain valuable, as in previous case. The same consideration about "time-locality" can be applied

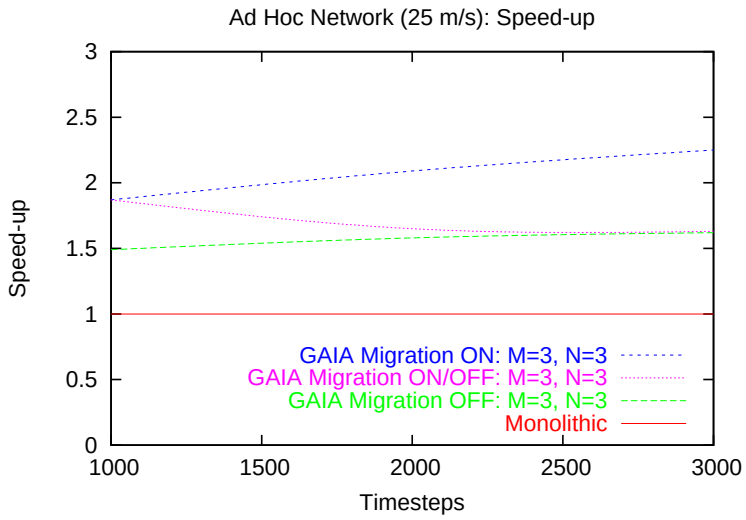

Figure 4.7: Transient speed-up effect over ARTÌS with and without the GAIA migration mechanism. Average SMH speed: 25 m/s.

to explain why the speed-up degradation when the GAIA migration is switched ON/OFF at runtime is faster in Figure 4.7 than in Figure 4.6.

Figure 4.8 shows the speed-up investigation of many execution system architectures, based on the mobile ad hoc network model characterized by 5000 SMHs with average speed of 25 m/s. Every bar in the histogram shows the speed-up with respect to the monolithic implementation, with GAIA Migration Off and On, respectively. The first couple of bars on the left are just a reference of the monolithic normalized speed-up. In general, the GAIA migration has a positive effect on the speed-up indices, with many PEUs, by increasing the speed-up indices up to 25%. More specifically, the second couple of bars shows the speed-up obtained by 3 LPs over 1 PEU. This indicates that ARTÌS is able to exploit the shared memory, dual processor architecture of the PEU, when implementing the simulation splitted on 3 LPs. In the same way, as reported on the histogram, by increasing the number of PEUs (M value) in the execution architecture, ARTÌS and GAIA show to scale and to support more LP executions by gaining simulation speed-up. In addition, in all the execution scenarios, GAIA dynamically recovers the overhead of network communication due to model assumptions (SMH mobility), resulting in additional

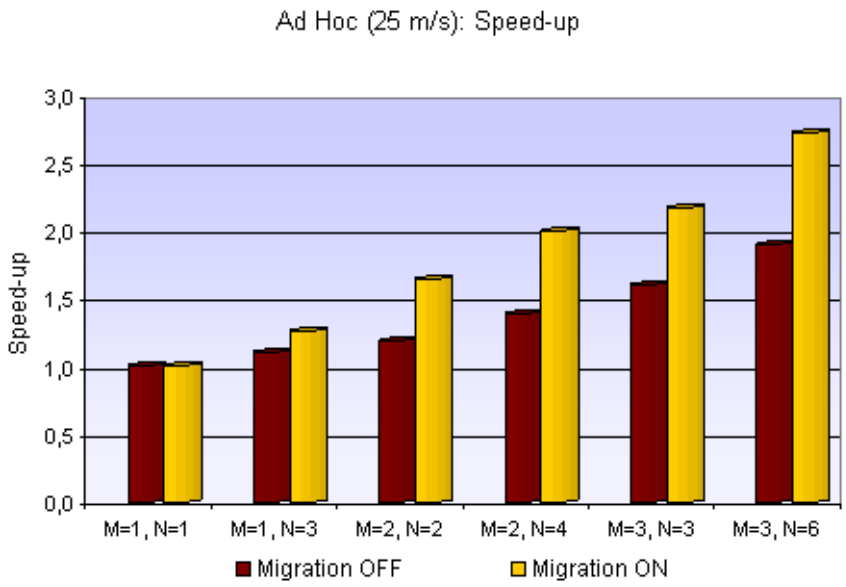

Figure 4.8: Speed-up investigation of ARTÌS and GAIA over many execution system architectures.

speed-up.

4.8 Sensor network's simulation

Figure 4.9 shows the scalability and speed-up obtained by the simulation of the sensor network model previously defined. The effect of GAIA here is not considered because we are testing the scalability of the model and simulation implementation for a model composed by static wireless sensors. Again, every PEU considered here is a shared memory, dual processor architecture. It was considered 3 execution scenarios. The monolithic scenario is realized by one LP over one PEU. The scenario labeled (M=3, N=6) is realized by 3 PEUs connected by the Ethernet LAN, each one with 2 LPs executed over the dual processor, shared memory architecture. The scenario labeled (M=2, N=4) is realized by 2 PEUs connected by the Ethernet LAN, each one with 2 LPs executed over the dual processor, shared memory architecture. The figure shows the speed-up obtained as a function of the number of modeled

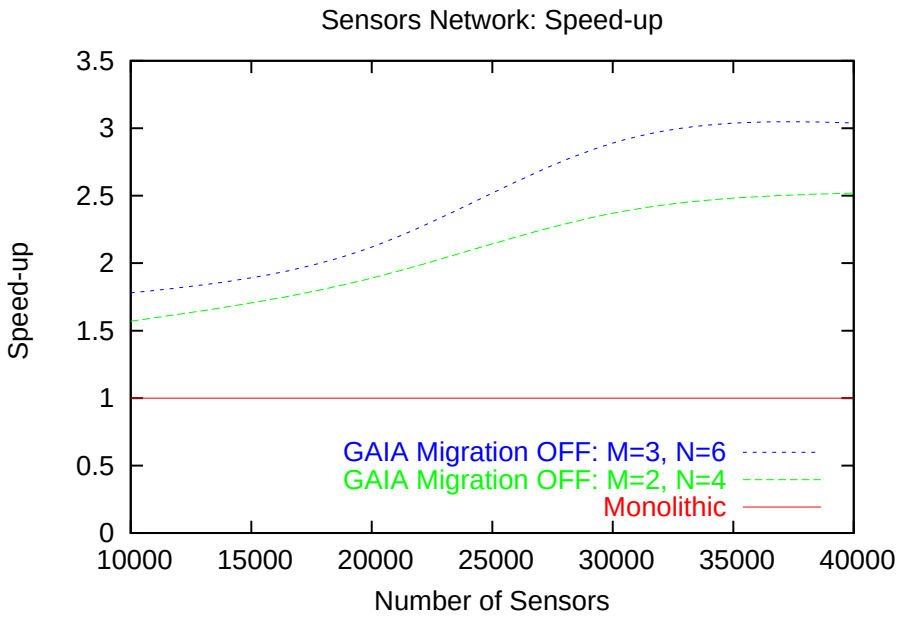

Figure 4.9: Speed-up and scalability investigation of ARTÌS for a massive sensor network model.

sensors, i.e. the ARTÌS speed-up and scalability under the model complexity viewpoint. The speed-up obtained increases with the number of simulated sensors. This can be explained because a huge number of sensors could exploit the potential for parallel computation expressed by the multiple number of processors in the execution architecture. The speed-up obtained by 3 PEU outperforms the speed-up obtained with only 2 PEUs, as expected. When the number of sensors is really high (i.e. around 40.000) the speed-up index reaches the top value, and does not show reductions, indicating the good scalability achieved by the simulator performance. As a marginal note, by considering that a state occupancy of a sensor model entity in our experiments was around 250 bytes, we executed a single experiment for a simulation of 1.000.000 sensors without having evidence of any problem. Additional investigation will be performed on the evaluation of GAIA reallocation mechanism under the sensor network scenario, in the initial phase. This would contribute to further optimize and increase the local communication and speed-up obtained.

4.9 Conclusions and future work

In this chapter has been presented a testbed evaluation of the ARTÌS middleware for the simulation of large scale wireless systems. Two classes of widely investigated wireless models were simulated: mobile ad hoc and static sensor networks. In the first case has been demonstrated as the integration of the GAIA framework ("simulated entities migration") leads to increased model scalability and speed-up. In the case of static sensor networks an optimal static allocation could be performed at bootstrap and so the GAIA framework is not required. In this case the middleware has proved to be scalable and we plan to further enlarge the number of simulated entities involved in our testbeds.

Our ongoing work includes the definition of new models for dynamically interacting systems like multi-agent systems, P2P models, scale free networks, complete protocol stacks for ad hoc and sensor models, biology-inspired models and molecular systems.

Chapter 5

Concurrent Replication of PADS

"Replicants are like any other machine. They're either a benefit or a hazard."
Ridley Scott, Blade Runner

Parallel and Distributed Simulations (PADS) enable the analysis of complex systems by concurrently exploiting the aggregate computation power and memory of clusters of execution units. In this chapter it will be investigated a new direction for increasing both the speed-up of a simulation process and the utilization of computation (and communication) resources. A typical implementation of a simulation-based investigation requires to collect many independent observations for a correct and significant statistical analysis of results. On the other hand, the execution of many independent parallel or distributed simulations may suffer the speed-up reduction due to rollbacks under the optimistic approach, and due to idle CPU times originated by synchronization and communication bottlenecks under the conservative approach. It will be introduced a parallel and distributed simulation framework supporting Concurrent Replication of Parallel and Distributed Simulations (CR-PADS), as an alternative to the execution of a linear sequence of multiple parallel or distributed simulation runs. This approach is substantially different from the Parallel and Distributed Simulation Cloning concept, and is quite different from the Multiple Replications in Parallel (MRIP) approach adopted for the concurrent execution of independent sequential simulation runs. The implementation of the CR-PADS mechanism has been defined and tested over the Advanced RTI System

(ARTÌS) framework. Results obtained from tests executed under variable scenarios show that speed-up gains could be obtained by adopting the proposed replication approach in addition to the pure parallel and distributed simulation.

5.1 Introduction

Many fields of research currently adopt simulation-based techniques for the analysis, in order to obtain deep insight of new systems‘ design, tuning and optimization. Many systems of interest for the analysis may be characterized by a complex model definition and dynamic interaction among a possibly huge set of model components. Complex dynamic systems, (as an example, like wireless networks, systems on chip, molecular systems), are of great interest in current research [39, 76, 115, 116, 117, 118, 130]. A complex dynamic system can be defined as a system whose model is composed by a high number of model components, and where the interactions (i.e. the causal effects of events) i) happens frequently among model components, and ii) they are dynamically subject to fast changes driven by the system (and model) evolution.

One of the main problems to deal with when implementing a parallel and distributed simulation is the communication and synchronization among the distributed simulation components (i.e. the federates). In general, an inverse tradeoff exists that determine a mutual worsening in the speed-up: i) by reducing the degree of parallelism in the computation or, conversely, ii) by the arising of communication bottlenecks and blocking synchronization primitives among many heterogeneous simulation components. A perfect load balancing among the execution units is difficult to obtain, due to the model dynamics and the asymmetry of the physical execution units. For these reasons, a majority of fast components usually may be idle waiting the synchronization of at least one single late component. This means that the execution of a majority of execution units is often blocked for significant time, and proceeds with the speed of the slowest component. The frequent synchronizations are usually implemented as message passing primitives, and may be

heavily affected by the arising of overheads for communication management and communication bottlenecks. In order to exploit the maximum level of computation parallelism, many research activities dealt with dynamic balancing of logical processes‘ executions (both cpu-loads and virtual time-advancing speeds) by trading-off communication, synchronization and speed-up, both in optimistic and conservative approaches[62, 65, 67, 79, 124, 79]. Many approaches for the implementation of parallel and distributed simulations have been investigated in order to reduce the overhead effects of distributed synchronization and communication in both optimistic and conservative distributed simulations. The motivation for the communication-reduction approach is the frequent adoption of networked clusters of PCs, in the place of shared-memory or tightly-coupled multiprocessors, as the execution units of the distributed simulation, primarily for cost reasons. The high network latency in these clusters could play a fundamental role in determining the weight of communication and synchronization between the distributed model components. Attempts have been performed in order to introduce adaptive behavior in the management of the model at runtime, to control the arising of overheads due to the model dynamics. Given the premises above, many results have demonstrated that a speed-up is obtainable from parallel and distributed simulations, under both the optimistic and conservative approaches, and for many system models and execution architectures.

In this chapter it is investigated a new direction for trying to maximize the speed-up of the simulation processes and the utilization of computation (and communication) resources in the system architecture supporting the parallel or distributed simulation. Our assumption, based on a common implementation rule, is that a simulation-based analysis is not limited to the execution of a single run of a parallel or distributed simulation, but requires many independent set of observations for a correct and significant statistical analysis of results. Our proposal is to realize a parallel and distributed simulation framework which is able to implement Concurrent Replications of Parallel and Distributed Simulations (CR-PADS), rather than executing a sequence of multiple parallel or distributed simulation runs. The replication concept [64, 70, 76, 92, 98] is intended here as a mechanism that duplicates

the logical processes (LPs) of parallel and distributed simulation runs starting from the initialization phase of every single run. Every replica is based on the same model definition, and realizes and independent execution based on local initial parameters, variable factors for the analysis, and different random number generation seeds. Even if the common aim in background is to obtain speed-up, this approach is substantially different from the simulation Cloning concept [57, 85, 87, 88, 89]. Simulation cloning has been demonstrated a good technique for supporting "faster than real time" simulation [86, 87, 88, 89] and "what if" analysis [57] being based on the active cloning of multiple instances of the initial simulation process at "decision points" that may be met during the simulation. In the CR-PADS approach the replication of parallel and distributed simulation processes is performed only by starting each replica from the beginning, and all the replicas do not necessarily need to be synchronized (i.e. they are independent under the time management viewpoint). The CR-PADS approach is not intended as a way to investigate all the possible evolutions from a "decision point", like in the cloning approach, but it is intended as a way to maximize the speed-up and utilization of system resources when implementing a set of parallel and distributed simulations of complex dynamic system models. Our Replication approach is also quite different from the Multiple Replications in Parallel (MRIP) approach adopted for the concurrent execution of independent sequential simulation runs, like in the Akaroa2 framework [64, 70, 76]. The aim of MRIP in Akaroa2 is basically to give a simple way to the modeler for initiating multiple independent runs of sequential simulators over different processors. It seems that Akaroa2 offers a controlled environment for launching multiple independent sequential simulations, each one executed over a single CPU, without managing the concept of parallel and distributed simulation. Akaroa2 incorporates runtime data analysis and data collection services to analyze transient and steady-state phases for each run, and to evaluate simulation ending conditions when the planned level of confidence is obtained by the set of results. With respect to CR-PADS approach, Akaroa2 and the MRIP approach may have limitations and could not exploit the aggregate resources of a cluster of execution units, and the adoption

of distributed models, which are some of the motivations in favor of the parallel and distributed simulation. The implementation of the CR-PADS mechanism has been defined and tested over the Advanced RTI System (ARTÌS) framework (Chapter 3).

Results obtained for the simulation of a complex dynamic system model (i.e. a wireless ad hoc network model) demonstrate that a speed-up gain can be obtained by adopting the proposed replication approach as an alternative to a sequence of standalone parallel and distributed simulations. The speed-up is obtained up to a given amount of replicas and has evidenced a dependence on the model characteristics. Basically, trashing effects are introduced when the saturation of the computation power of all the CPUs has been achieved. An excessive number of replicas would result in additional trashing effects under the management viewpoint. Given the light and efficient implementation of CR-PADS the number of replicas causing trashing is in a range that exceeds the typical amount of runs required to achieve a good statistical relevance, i.e. thin confidence intervals, from the collected observations.

The chapter structure is the following: in Section 5.2 are outlined some concepts about the parallel and distributed simulation cloning and replication; in Section 5.3 the key issues for the replication mechanism implementation and the ARTÌS middleware are defined; in Section 5.4 a prototype wireless system‘s model and a set of simulation results are presented to evaluate the concurrent replication approach; in Section 5.5 conclusions are summarized.

5.2 Cloning and replication of PADS

5.2.1 MRIP

The Multiple Replications in Parallel (MRIP) technique is a technique that will be cited here as a reference for this work [76, 70, 76, 92, 98]. This technique consists in launching multiple runs of independent sequential simulations in parallel over a set of concurrent CPUs. Every simulation run is executed from the beginning up to the end on the same CPU, under the control of a single scheduler (i.e. a

monolithic sequential simulation). Some frameworks like Akaroa2 provide support for the MRIP when launching simulations based on common sequential simulation tools like PTolemy, NS2, OMNET++ [34, 32]. To the best of our knowledge, the MRIP approach for parallel and distributed simulations is still to be investigated.

5.2.2 Parallel and distributed simulation

The architecture of the physical execution units (PEUs) can be organized in different ways: from a parallel multi-processor architecture with shared memory up to a distributed cluster of PCs interconnected by LANs or even by the Internet. Historically, the research community called a parallel simulation the concurrent execution of a single simulation run over a tightly coupled multi-processor architecture, and a distributed simulation the concurrent execution of a single simulation run over a loosely coupled set of execution units, each one running on a possibly different HW architecture and separate local memory. In the optimistic approaches for implementing parallel and distributed simulations, several simulation components may bet on forecasting and computing one out of the possible evolutions, in order to obtain a maximum exploitation of the parallel execution. In solutions like the Time Warp [91], many model components advance their evolution without worrying about causality maintenance at least until a violation of causality is revealed: in such case a costly process called rollback is executed to restore the processes states to a global safe-state. Such optimistic implementations were thought as a way to maximize the utilization of expensive computation architectures. The efficiency of optimistic implementations would depend heavily on the evolutionary characteristics of the models: highly independent, predictable or self-correlated models would behave better than unpredictable ones, since independent sub-models may have few common "decision points" affecting each others' evolution, and the choices made at "decision points" could be performed on the basis of more effective "oracles" that could reduce the need for frequent rollbacks. Needless to say that rollbacks can reduce the efficiency of the simulation process in significant way. In the conservative approach, each "time advance" in the model evolution is made under the conserva-

tive assumption that all previous events have been processed in correct timestamp order, by all the parallel and distributed model components. Frequent synchronizations (i.e. blocking and unblocking event executions) of all the model components are performed for ensuring a conservative implementation of the causal order of events.

5.2.3 IPC communication for parallel and distributed simulation

The communication among LPs in a parallel simulation is usually efficient and reliable because it can be supported by the classical mechanisms for local inter-process communication (IPC) like pipes, FIFO channels and Shared Memory. Possible advantages given by local IPC communication mechanisms are: reliable communications, ordering maintenance of messages (excepted shared memory) and efficiency, intended as high bitrate and low latency channels among LPs. The shared memory solution requires a control of the concurrent accesses to mutual exclusive areas: the efficient implementation of control primitives by the operating system is a necessary condition for the efficiency of the communication. The problem with the parallel simulation approach is given by the scalability of the model and simulation. In other words, the physical execution architecture may be limiting the possible number of concurrent LPs that can be executed by maintaining the advantages of concurrent computation. The distributed simulation approach is based on the implementation of concurrent LPs executed over distributed physical execution units (PEUs). The advantages of distributed simulation architectures can be summarized as: i) theoretical scalability, given by the arbitrary extension of the PEU architecture, ii) the possible geographical distribution of PEUs, which could be exploited to deal with management and reliability issues, and iii) the fault tolerance based on the possible substitution of unreliable or disconnected PEUs. The communication among LPs can be supported typically by external inter-process communication mechanisms, i.e. message passing based on packet-based communication over heterogeneous in-

terconnection networks. External IPC solutions are usually less reliable and efficient than local IPC. The assumptions about the network infrastructures to be adopted ranges from efficient LANs up to unreliable and high-latency Internet-based communication. In some scenarios, both parallel and distributed PEUs can be used to execute simulations. Given the low performance and reliability of network based communication, it is self-evident that local IPC is preferable (if available) to be exploited over parallel architectures.

5.2.4 Parallel and distributed simulation cloning

The simulation cloning technology was introduced as a concurrent evaluation mechanism, in the context of parallel simulation [85]. Simulation cloning allows the creation of copies of a simulation process (clones) at "decision points", which are evaluated at runtime, but need to be defined preliminary in the model design phase. When a set of clones is created, each clone would execute a different possible evolution of the current scenario, each one related to any choice made at the decision point. The main flow of events of a simulation may be recursively split in separate flows characterizing a different evolution of clones, starting from the decision points. The advantage of simulation cloning is basically obtained by the concurrent investigation of alternative choices, by exploiting the concurrent computation of parallel and distributed architectures. Clones‘ evolutions originated by bad choices can be killed at runtime to reduce the overheads. This technique was motivated by the need to develop a parallel model of execution that supports an efficient, simple, and effective way to evaluate and compare alternative scenarios, originated from a common point in the model evolution. Parallel discrete event simulation can adopt the cloning concept on the basis of the logical processes (LPs) execution model. In [119] the cloning approach was designed for a more flexible system composition. The cloning approach includes the management of different time axes in parallel, in order to support a runtime forecasting functionality. Internal cloning and external cloning techniques were suggested to clone the federates at runtime. In [57] the aim to support users willing to run existing complex simulation models, gave reusability

and transparency issues a top role while enabling simulation cloning. This is the reason for cloning design on HLA-compliant distributed simulations, which led to the introduction of the concept of virtual federates [57, 87]. All the works in the literature based on simulation cloning principles have their own motivations and relevance, and illustrate interesting issues and solutions. Specifically, cloning was demonstrated as a valuable technique for supporting "faster than real time" and "what if" decision processes. The parallel and distributed simulation guidelines applied to the cloning are interesting, because they allow to exploit the concept of virtual logical processes (VLPs). Each simulation clone can be implemented as a set of VLPs communicating by using virtual messages (VMs). The management of VLPs and VMs allows an overhead reduction in the communication among LPs, because many VLPs can be associated to a single LP, and many VMs can be included in one real message exchanged among LPs. This approach was demonstrated to reduce the overheads in the communication and execution management of parallel and distributed simulation clones [57, 87, 88, 89]. More specifically, in the management of clones' executions, a good runtime implementation would allow to determine which LPs have diverging executions and which haven't, in order to reduce the redundancy of LP copies and LP messages. Needless to say that the design and implementation of runtime support for cloning of HLA-based parallel and distributed simulations is quite complex and challenging effort. A preliminary discussion of the design and implementation challenges and solutions can be found in [57, 87, 88, 89]. On the other hand, the cloning technique can be considered a good technique for speed-up purposes, but previous implementation works demonstrated that the cloning implementation requires a complex management. This is even more critical under HLA-based and Data Distribution Management (DDM) based implementations. The specific benefits of simulation cloning in the analysis of "faster than real time" and "what if" analysis, and the complex management required, motivated our effort to exploit the replication concept of LPs with a more simple approach.

5.2.5 The concurrent replication of parallel and distributed simulations

The replication concept described in [70, 76, 92, 98] is intended here a mechanism that duplicates the logical processes (LPs) of parallel and distributed simulation runs starting from the initialization phase of every single run. Every replica is based on the same model definition, and realizes and independent execution based on local initial parameters, variable factors for the analysis, and different random number generation seeds. In other words, many independent simulation runs are executed concurrently by replicating them (just as clones) only at the beginning of the simulation process. Each replica is an independent run, with its own seeds, and model initialization. The implementation of the concept of replication of HLA-based parallel and distributed simulations inherits some of the guidelines from the cloning design, but also have a more clean and simple management.

Motivations for the concurrent replication

In the following will be explained the motivations for the proposal of a Concurrent Replication mechanism for Parallel and Distributed Simulation (CR-PADS) by sketching the differences among the MRIP, the PDES and the CR-PADS approaches, under the computation and communication viewpoints.

By assuming that a set of N CPUs are made available for computation (no matter if they are belonging to parallel or distributed architectures) we want a set of many independent runs to be executed. Obviously, the aim is to have the completion of the overall simulation process in the lowest time and with the maximum utilization of computation and communication resources. By focusing on the MRIP approach the independent runs can be executed by launching in parallel the sequential simulations over the available CPUs. It results that the possible concurrency of the model execution cannot be exploited to obtain simulation speed-up, because every computation is linearly executed as a sequence of tasks. In this scenario, the model data structures and computation must fit on the CPU system and may suffer memory

and computation limitations. Moreover if the available resources are more than the number of runs required, the potential associated to some resources may remain not exploited. The parallel (or distributed) discrete event simulation (PDES) approach may introduce advantages, because every independent run could exploit the whole computation architecture, by mapping and exploiting the degree of parallelism inherent to the model over the concurrent CPUs. This implies that a single run may complete in less time than a sequential run. On the other hand, the linear execution of two (or many) runs may result in a speed-up depending on the number of resources and the number of runs required under MRIP and PDES, respectively. It is worth noting that the advantages of the aggregate memory architecture may assist the model data structures management, and that the whole set of computation resources (CPUs) can be exploited in parallel. Under the PDES scenario frequent synchronizations are required among the model components (by assuming a conservative event-based or time-stepped implementation). Every synchronization barrier initially unblocks the concurrent computation of CPUs. As soon as the computation phase is terminated, every process starts a message passing phase to synchronize again its execution with other processes. This implies that i) the whole set of processes advance with the speed of the slowest (or more computation intensive) one, and ii) the final phase before the synchronization barrier is communication-intensive and may suffer additional delays due to the congestion and delays of the inter-process communication infrastructure. The communication delay problem may result in a high percentage of synchronization delay, under loosely coupled distributed architectures (e.g. over CPUs interconnected by LAN or Internet technology). In other words, between any couple of synchronizations, every CPU swings between computation and idle periods, while the underlying communication infrastructures swings between idle and communication periods, respectively. The interesting question we want to investigate in this work is: could we try to obtain a more fluent computation and communication by merging the execution tasks of more than one parallel or distributed simulation replica over the computation and communication architecture? In other words, by launching multiple, independent and concurrent parallel

or distributed simulation runs over the system we may obtain that CPUs do not spend so much idle time waiting for synchronizations because they can switch to the execution of computation requests by the other replicas which already completed their synchronization phase. The same happens under the communication system viewpoint, because the message passing from all the replicas may increase the uniform utilization of the communication system. As a result, idle CPU times and idle or congested communication channels could be obtained in a more smoothed way, and this may result in additional speed-up with respect to the whole time required for completing the set of simulation runs. The risk in this approach is to spend too much time in switching processes‘ executions, and in the creation of communication bottlenecks and livelocks, resulting in trashing executions. Our design is based on a set of guidelines that we followed in order to obtain the maximum advantage from the replication mechanism, by opportunely managing the processes executions and communications, and by keeping under control the overheads introduced.

Advantages of the replication approach

Among the advantages of a concurrent replication approach for parallel and distributed simulations, at the top layer can be identified the possible speed-up obtainable when executing a set of many independent runs, or the possible concurrent analysis of different scenarios (defined at the beginning of the run). The implementation guidelines, partially inherited by the existing design of the cloning approach, and partially object of our research, can result in additional advantages. As an example, the structure of the ARTÌS framework has been defined such that a separation of the simulation and replication management is specifically oriented to a clean design and to the exploitation of management techniques that reduce the communication overheads.

Our choice is to create replicas by replicating virtual LPs that realize a simulation run. A set of replicas of virtual LPs is managed as a single LP by the runtime management. This simplifies the management under the ARTÌS viewpoint and allows an optimization and balancing of the utilization of communication resources,

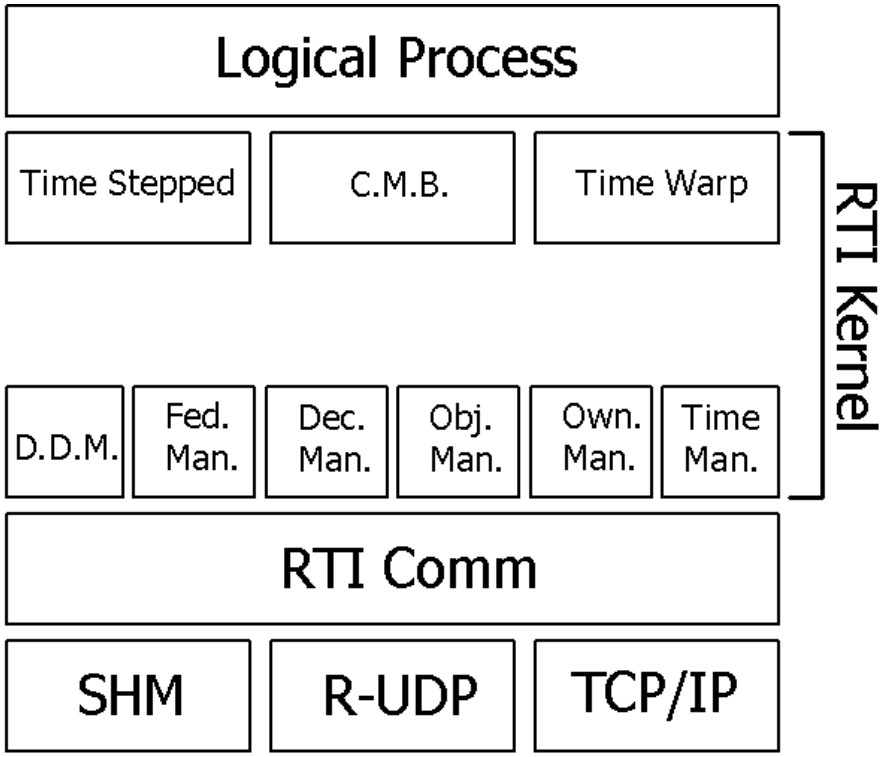

Figure 5.1: The ARTÌS architecture.

based on queue management, priority and fairness protocols. The management of Random Numbers Generators (RNGs) is simple, because seeds can be chosen at the beginning of the runs, without originating correlated sequences whose effect could bias the analysis of results at the end of a set of simulation runs. This effect is quite hard to manage in the cloning approach, since a set of clones always suffer a correlation effect due to the first common subsequence of random numbers used.

5.3 The CR-PADS implementation

5.3.1 Implementation of replication in ARTÌS

The ARTÌS logical structure has been detailed in Chapter 3, given the top level structure shown in Figure 5.1, ARTÌS supports the execution of LPs over the RTI kernel. The RTI kernel implements time management policies, object ownership and

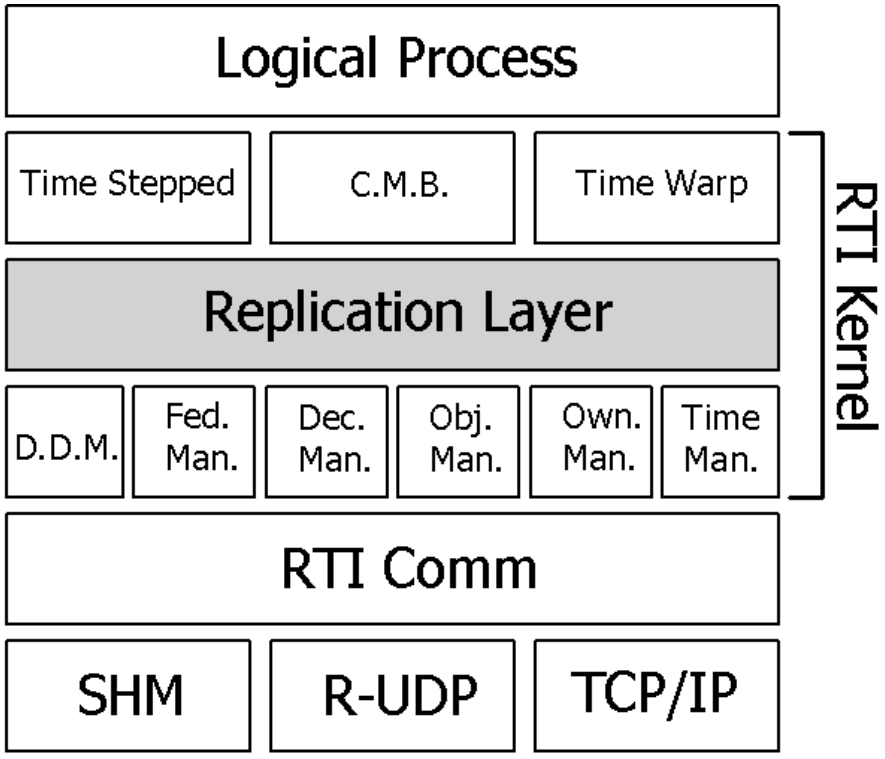

Figure 5.2: The ARTÌS and Replication architecture.

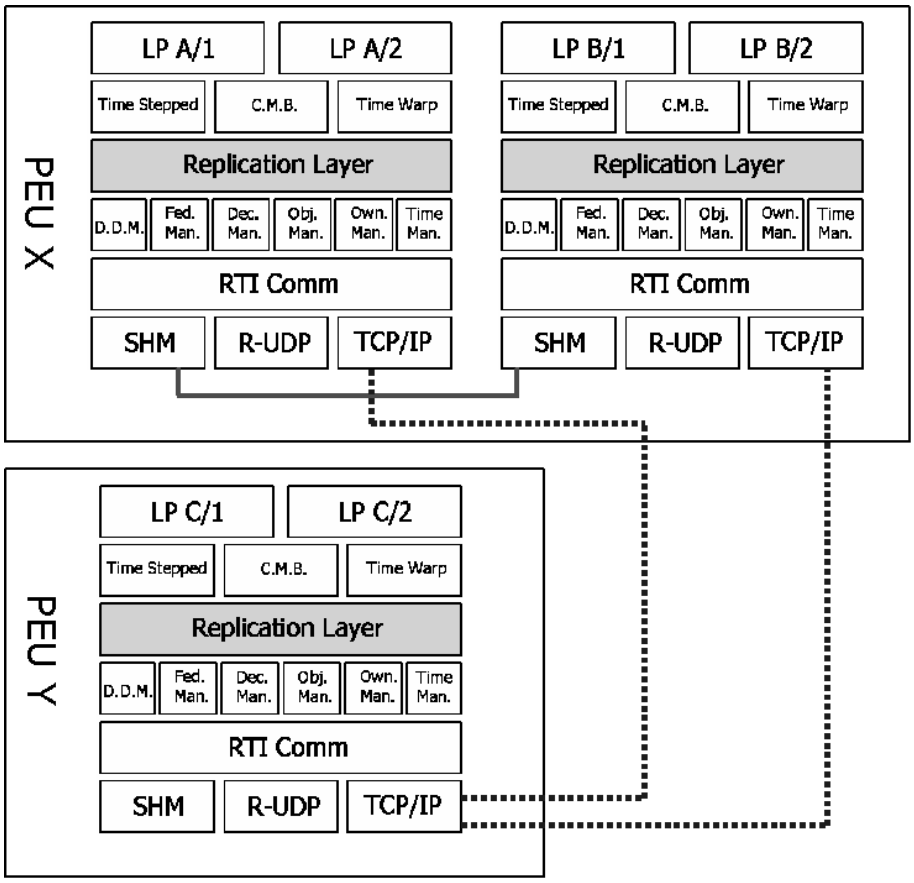

Figure 5.3: Parallel and distributed CR-PADS architecture.

declaration management, data distribution management, federation management and other functions. At the bottom layer of the RTI, the Runtime Communication layer (RTIComm) manages the communication based on the underlying communication system that connects the PEUs‘ architecture. Given the ARTÌS design, the abstractions of LPs, messages, channels and simulation-runs appear as objects implemented over the RTI kernel. In order to manage efficiently the management of messages among LPs, an active thread executes the task of being waiting for messages on each channel, and demultiplexing messages to the above layers by adopting an efficient callback mechanism. The time management layer also adopts callback techniques to avoid polling techniques over the RTIComm layer, that may reduce performances.

The Replication mechanism has been inserted in ARTÌS directly over the RTIComm layer. This facilitates the need to maintain transparency to the LPs, and light implementation of the replication mechanism. The choice to realize replication at the above layer would have given benefits under the optimization viewpoint, at the additional expenses of re-implementing ad hoc the replication services for each time management policy. This means that our approach is simple even if it does not provide space for optimizations like in the cloning approaches. On the other hand, given the straight-forward adoption of replication services, and the assumption about the independence of replicas, the need for replicas optimization management (e.g. deleting redundant messages) has been considered a secondary factor in the design.

The Replication management layer is the funnel for replicas over the RTIComm layer (see Figure 5.2. The Replication layer generates the replicas of each LP, and manages (that is, multiplexes and demultiplexes) messages from/to LPs of each replica. The set of processes and threads of each simulation replica has been designed as a tree-like structure, whose inter-process communication is based on highly efficient UNIX pipes [32]. UNIX pipes have been preferred to shared data structures, because messages have limited size and shared data structures consistency management would introduce latency and additional overheads. In the following, we illustrate the dynamic behavior of the Replication management layer with respect to the

replication of a LP in a parallel or distributed simulation run.

1) the user‘s simulator code is compiled with ARTÌS
2) the simulation run is started
a) execute initialization
b) call ARTIS_init() API who creates LP_father
3) ARTIS_init() calls Replication_init()
4) LP_father calls RTI_kernel_init()
a) RTI_kernel_init() creates threads for communication and initialization of LP replicas that will be created by the LP_forker thread
b) each LP replica create a new pipe_listener thread to receive messages from the LP_father
5) the LP_father creates the LP_forker process
a) LP_forker process generates new replicas (upon request from the user)
6) LP_father waits on its own communication pipes
7) Simulation run starts
a) once LP replicas send messages, the Replication layer sends the message to the LP_father pipe
b) once the LP_father receives messages, it adds headers for multiplexing and manage the message by adopting the standard RTI kernel functions to send the message to the receiver LP.
c) once a message is received by the thread of the receiver LP, the receiver LP demultiplexes the message to the pipes of the LP replicas, where the message is handled

The set of operations 7.a.b.c is executed until the LP process terminates, and all its resources are released.

5.4 Performance evaluation

To test the proposed framework was implemented a time-stepped, conservative, parallel and distributed discrete-event simulation of a mobile wireless system.

5.4.1 Simulation system and simulation model

Our simulation testbed consists of two different environments: i) parallel and ii) distributed discrete-event simulation of model components. In the parallel environment for simulation the model components are executed as logical processes over a dual processor physical execution unit (PEU). Specifically, the PEU is an Intel Dual Xeon Pentium IV 2800 Mhz, with 3 GB RAM, Debian GNU/Linux OS with kernel version 2.6. In the distributed environment the logical processes are mapped over a set of physical execution units (PEUs), connected by a physical LAN network (Fast Ethernet 100 Mb/s).

As a testbed for the replication framework, was realized a conservative, time-stepped simulation of a complex and dynamic model. The simulation model we considered for the simulations is a wireless mobile ad hoc network model. The mobile ad hoc network is realized by Simulated Mobile Hosts (SMHs), each one characterized by random mobility and CBR traffic (that is, ping messages sent to all the neighbor SMHs in their reception area). The number of simulated SMHs as the effect of control the average SMH density in the system. The SMH mobility causes changes in the network topology and the SMH dynamics. Since the model design is out of scope in this thesis, I'll simply characterize the model execution by noting that: i) the computation required for each SMHs per timestep is in the order of O($\#SMH^2$) and, ii) the communication and synchronization required among SMHs is in the order of O($K * \#SMHs$) per timestep. All the model choices have been defined in order to realize a stressing test for our simulation framework.

5.4.2 Performance results

In this section will be presented the results of some testbed simulation experiments executed to analyze the performance of the proposed CR-PADS approach in presence of parallel and distributed environments.

It were performed multiple runs of each experiment, and the confidence intervals obtained with a 95% confidence level are lower than 5% the average value of the

performance indices shown.

In the following will be defined M as the number of physical execution units (PEUs) supporting the simulation execution, and N as the total number of logical processes (LPs) implemented for each simulation run. As mentioned above, each PEU is composed by a dual processor machine. With the "CR-PADS OFF" label we identify a legacy parallel and distributed simulation approach: that is, simulation runs are executed in sequential order, by activating N LPs at a time. With "CR-PADS ON" we identify a parallel or distributed simulation executed under the effect of the CR-PADS replication approach described in previous sections. The "number of replications" is intended as the number of independent simulation runs executed (both sequentially when CR-PADS is OFF, and in concurrent way when CR-PADS is ON).

In Figures 5.4, 5.5 and 5.6, will be reported results for the parallel simulation environment (M=1, N=2). Each figure shows the Time (WCT) required for completing the simulation processes. Figure 5.4 shows the effect of 500 SMHs involved in the simulation (250 SMHs over the N=2 LPs executed over the M=1 dual processor PEU). Figure 5.5 and 5.6 show the same index with 1000 and 2000 SMHs, respectively. Results confirm that the CR-PADS approach outperforms the sequential approach by considering the WCT required to complete a whole set of simulation runs. Each run is defined with fixed size of 300 timesteps. In Figure 5.4, by increasing the number of concurrent replications (up to 20), the CR-PADS results are better than the traditional parallel simulation approach. By increasing the number of SMHs up to 1000 (Figure 5.5) and 2000 (Figure 5.6) we are increasing the percentage weight of local computation, with respect to the percentage weight of communication, in the sequence of synchronization steps, for each SMH during the simulation runs. This means that we are pushing the computation to saturate all the CPUs computation power. When the computation load for LPs asymptotically saturates the CPU computation power, the CR-PADS approach becomes less efficient than the legacy PADS approach, because it does not longer exploit any idle CPU time in order to execute more concurrent replicas. It is worth noting that

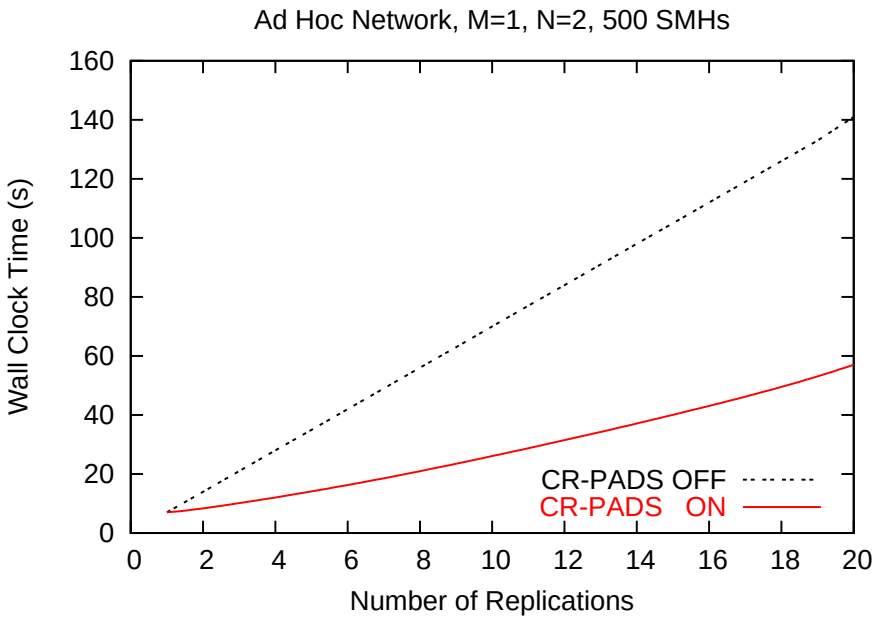

Figure 5.4: total WCT vs. Number of runs (replications) Parallel simulation scenario: M=1, N=2, 500 SMHs.

the break-even in the number of replicas is around 15 in Figure 5.5 and around 6 concurrent replicas in Figure 5.6.

As it was expected, the CR-PADS approach introduces overheads when the number of replicas is high and the computation of few replicas saturates the computation power of all PEUs. It is worth noting that in all the proposed scenarios, the CR-PADS approach can give a speed-up effect, at least limited to the initial subrange in the number of concurrent replicas.

By focusing our attention to the distributed environment (with M=3 and N=6), in Figures 5.7, 5.8 and 5.9 are illustrated the same results shown for the parallel simulation scenario. It is worth noting that the average communication latency of distributed environments is at least one order of magnitude bigger than the average latency experienced in the parallel architecture. A significant latency implies long time required to achieve synchronization at every timestep. This would translate in better opportunities for CR-PADS to optimize the concurrent execution of many runs, by exploiting a better utilization of the PEU computation power, and by

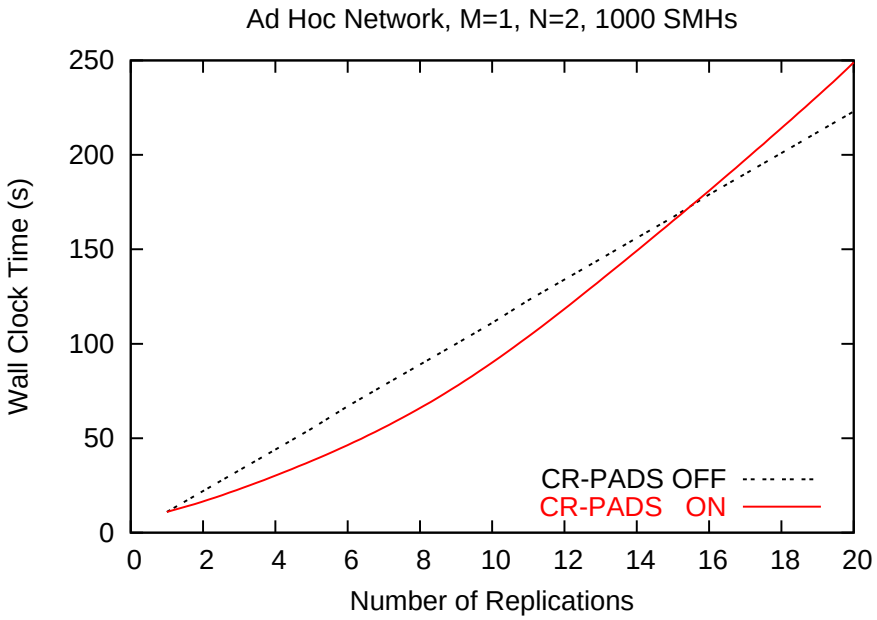

Figure 5.5: Total WCT vs. Number of runs (replications) Parallel simulation scenario: M=1, N=2, 1000 SMHs.

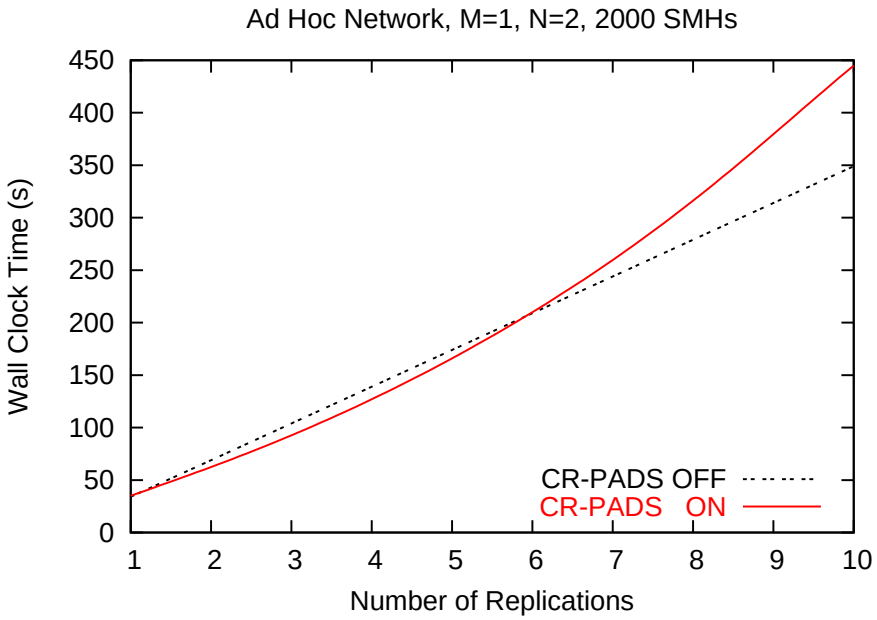

Figure 5.6: Total WCT vs. Number of runs (replications) Parallel simulation scenario: M=1, N=2, 2000 SMHs.

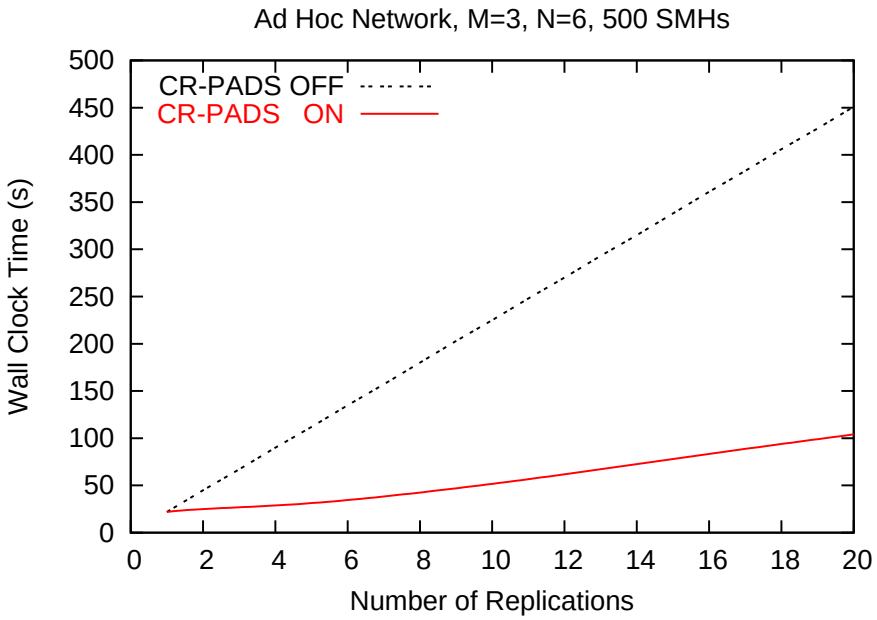

Figure 5.7: Total WCT vs. Number of runs (replications) Distributed simulation scenario: M=3, N=6, 500 SMHs.

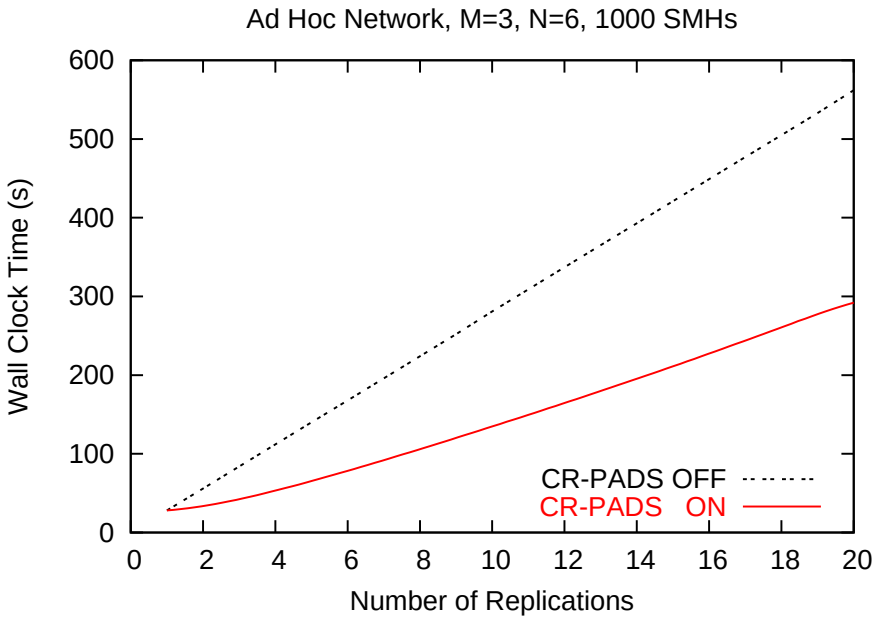

Figure 5.8: Total WCT vs. Number of runs (replications) Distributed simulation scenario: M=3, N=6, 1000 SMHs.

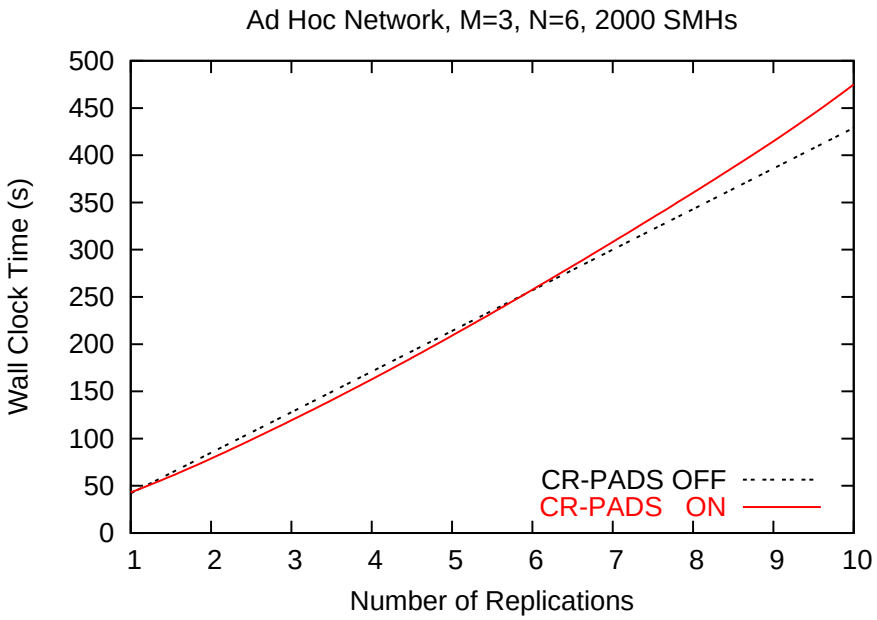

Figure 5.9: Total WCT vs. Number of runs (replications) Distributed simulation scenario: M=3, N=6, 2000 SMHs.

reducing the global WCT required for completing the simulations. The results confirm the expectations: for a number of concurrent runs ranging from 1 up to 20, the CR-PADS mechanism is able to give significant speed-up both in the 500 and 1000 SMHs scenarios (Figures 5.7 and 5.8). When the scenario becomes computation-intensive (2000 SMHs in figure 9) the CR-PADS approximates the same results than the legacy PADS approach, and do not introduce significant overheads in the range of 1 up to 10 concurrent replications.

Figure 5.10 illustrates the rate of event processing under both parallel and distributed scenarios obtained for 10 simulation runs executed with and without the CR-PADS framework in background. By looking at the Figure 5.10, CR-PADS allows a high event computation density when the computation load in each timestep do not saturate the available CPUs (that is, when SMH=500..1000). When the computation load increases (SMH=2000) the trashing effect of CR-PADS appears, basically because there is not space for additional concurrency in the computation. In the distributed scenario, the high latency of communication reduces the compu-

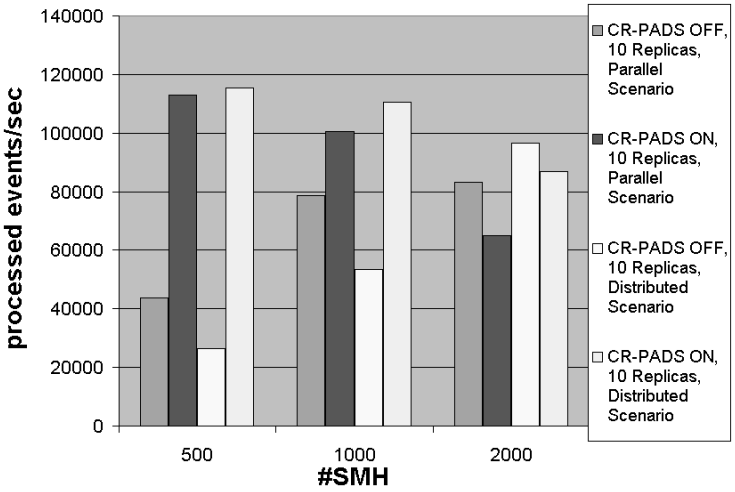

Figure 5.10: Analysis of event processing rate.

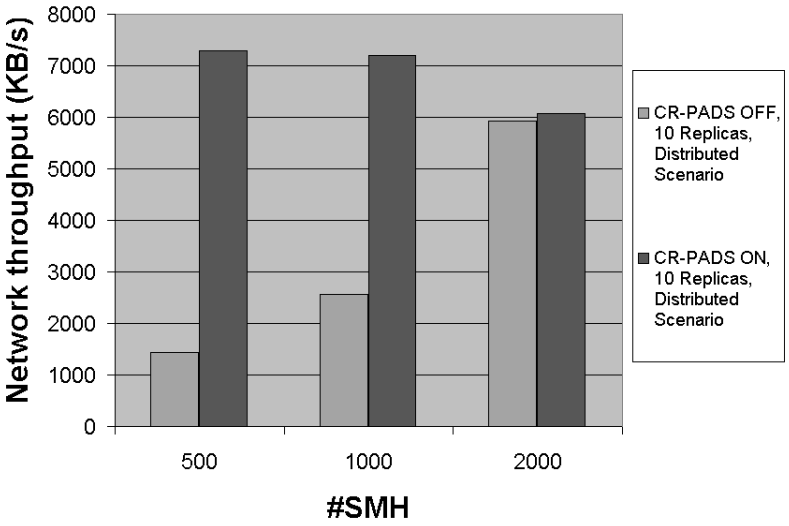

Figure 5.11: Analysis of network communication throughput.

tation concurrency in legacy distributed simulations. As expected, the CR-PADS mechanism is able to maintain a high computation concurrency. Figure 5.11 shows the effect of the communication layer during the execution of distributed simulations. It is clear how the CR-PADS mechanism is able to increase the throughput of communication channels even when the computation load is low. Conversely, when CR-PADS is off, the communication channels are under-utilized. When the computation load asymptotically saturates the available CPUs, both the CR-PADS On and CR-PADS Off implementations converges to the same network utilization. This is due because the network communication is generated as a function of the events processed, that is the computation power, which is the current system bottleneck.

5.5 Conclusions and future work

In this chapter has been proposed and investigated a new direction for increasing the speed-up of a parallel or distributed simulation. This result has been possible thanks to a better utilization of CPU and communication resources. Since a typical simulation-based investigation requires to collect many independent observations for a correct and significant statistical analysis of the results, we think that our approach could be really valuable in the implementation of simulators.

Our future work will include the optimization of the proposed framework, and the investigation of adaptive automation of concurrent replication. We plan to investigate the adoption of CR-PADS under other conservative and optimistic approaches for parallel and distributed simulation, and under massive parallel computation architectures. We also plan to integrate CR-PADS with a framework for adaptive load balancing and migration of simulated entities (GAIA), and with the components for runtime transient and steady-state analysis of data, confidence interval estimation, and run termination management.

Chapter 6

A Migration-based Architecture for Internet Games

> *"Computer games don't affect kids, I mean if Pac-Man affected us as kids, we'd all be running around in darkened rooms, munching magic pills and listening to repetitive music."*
> Kristian Wilson, Nintendo, Inc, 1989[1]

Traditionally the simulation community has been composed by at least three sub-communities of primary relevance: high speed computing, military (i.e. Digital Virtual Environments) and gaming. The last, year by year, has increased its importance due to the high market impact of the gaming industry. Under the technical viewpoint the Internet Gaming field has a lot to share with the parallel and distributed simulation background. In a lot of cases a game can be referred as a simulation with relaxed temporal and correctness constrains. With different degrees of relevance, in both cases, synchronization, fault-tolerance, performance and the support of massively populated scenarios are a relevant subset of the main problems to understand and address. Given the depicted technical situation and the requirements, the goal of the work described in this chapter is to leverage from the existing simulation technology to improve the the software architectures usually built to support massively populated Internet Games. The simulated entities'

[1] A few years later appeared rave parties, techno music and ecstasy...

migration concept (previously presented in Chapter 4.4) will be revised to improve overall performances but also to obtain a better "fairness" in the gaming evolution.

In recent years many popular interactive computer games have gained online remote multiplayer functionalities, supported by standard Internet communication protocols and architectures. Due to the heterogeneous communication infrastructures and network asymmetries, some users (i.e. clients) may be suffering slow, congested and unreliable Internet connections, while others may have access to fast and reliable links. A different rate and latency in the delivery of users' commands and event notifications may lead to unfairness issues during the game play, specifically, for the class of real time and interactive games. A dynamic adaptation of the gaming architecture to the limitations of the communication infrastructure could be exploited to reduce these problems. The communication network topology and performance should be considered in the management of the client allocation to the set of multiple servers. In this chapter is presented a simple client migration algorithm which can be adopted on a generic multiplayer, multi-server online gaming architecture. Client migration among the servers of the gaming infrastructure is exploited to adapt to the dynamic performances of the general communication network infrastructure. The proposed mechanism has been modeled and simulated for the class of distributed multiplayer and multi-server interactive games, implemented over a general communication network infrastructure. Results show a significant fairness improvement, more homogeneous performances, and the absence of significant overheads.

6.1 Introduction

The explosive growth of the Internet, and the even widely deployed access and mobile connectivity to the Internet infrastructure and services, motivated the designers of computer-games to create new, interactive, and distributed multiplayer games, or even to add online multiplayer capabilities to existing ones [44]. The aim of an interactive multiplayer game is to allow a set of players' avatars to interact in a virtual

environment, by following specific interaction schemes and game rules. Avatars are implemented as client applications coordinated and managed by a set of distributed servers. The gaming infrastructure is realized by the set of servers, the distributed client applications, the interaction rules and game management protocols, and the interconnecting communication network infrastructure. In simple interactive gaming infrastructures, the interaction policy may be straight-forward: e.g in turn-based games, every client gets its own turn to 'move" and every other client would see the action as the expected interaction event (like in a multiplayer cards game). This gaming infrastructure does not have critical network and fairness requirements [126], because every user waits its turn until other users performed their own actions in a predefined order. The more challenging and attractive set of today's multiplayer games lets the users to live and act by proactively generating local and personal actions (i.e. events), which may affect every other user within the game context [43, 81]. The density and the timing constraints of interactions among the dynamic set of clients, obviously depends on the game type and characteristics. It is worth noting that massive, highly interactive games may be critical to be supported in a distributed way, under the communication viewpoint. In this chapter the focus is on the class of highly interactive distributed multiplayer games, where each user may take any decision at anytime (i.e. with no predefined turns), like in a virtual battlefield game. For this class of games, the communication protocols, the gaming infrastructure, and the communication infrastructure are far more interesting and challenging to be designed. The main problem faced in this work is the heterogeneity and dynamicity in the latency that could exist on the Internet connections among many peer-players connected to the distributed multi-servers architecture supporting the game execution.

6.1.1 Work motivation

Many multiplayer real-time games (e.g. Ultima Online [110]) require that the player sends 'commands" (e.g. walk, take objects, fight, shoot) to a virtual character (i.e. the avatar). Commands are messages sent to one server, over the communication

infrastructure (i.e. the network), in order to be evaluated and executed. Event execution generates causally dependent actions and environment changes, to be immediately notified to other characters in the gaming environment. Failing to send a command or an event message, or sending it too late, may cause damages to the avatar evolution (death, injury, loss of resources), and may result in unjustified penalties for the player. For this reason, users with high-latency or low-bandwidth Internet connections may be more disadvantaged than their opponents supported by fast connections. The unfairness problem frequently leads to a generalized users' annoyance: slow users get frustrated because they lose easily despite their skills [83, 109], while fast (and fair) users get bored because they win easily thanks to their network performances. This would translate in less users buying the game and subscribing the online gaming services.

The communication protocols adopted for these games usually do not offer specific solutions for this problem. On the other hand, a general conservative approach is to send less data to slow users [123], thus lowering the game detail. The solution to achieve fairness by reducing the game detail could be unpractical and questionable, under the player satisfaction viewpoint. Other gaming infrastructures could obtain an improved fairness by exploiting network services implemented below the application, without having to touch the internal details of the game protocol. On the other hand, this would require a complete management of the Quality of Service over end-to-end connections, which is not currently implemented and supported on the Internet.

My proposal, described in this chapter follows a complementary approach. It is assumed a common network scenario, an online (Internet-based) multiplayer real-time game supported by a distributed multi-server gaming architecture. I argue that the fairness may be improved by acting on the dynamic migration of clients among servers, with minor changes to the game code, and without requiring to rewrite and redesign any communication protocol. Moreover, it is not assumed any specific gaming protocol semantics, because the interest is in proposing and evaluating a mechanism that could be adapted to any generalized gaming architecture

[80, 121]. In addition to the previous considerations, another motivation for the proposed migration-based approach is explained in the following. Let us assume that upon the initial connection of a new player to the gaming infrastructure, the client application is connected to the server with the best tradeoff between network performance and server load, among the available ones. Many policies could be defined to realize a ranking evaluation of available servers. Unfortunately, due to the highly dynamic characteristics of server loads and network performances, any optimal initial allocation would soon become sub-optimal. As an example, network loads, link utilization and router congestion over generalized multi-purpose networks are subject to fast and unpredictable changes, so that any optimal choice at time t may become suboptimal after few seconds. In practice, the proposal is to let the clients, being connected to one server, to *migrate* towards the "most performant" server evaluated at runtime. This migration should be evaluated on the basis of server-measurement metrics and heuristic policies. Also, the density of migrations should be controlled, as an example, by performing migrations only at predefined time-interval boundaries, or after some critical events occurred. In this way, any client should pursue the most performant server, by dynamically measuring the network and server performances. The performance estimates and the migration heuristic should be defined in order to avoid overheads and useless fluctuations of clients.

This chapter is structured as follows. In Section 6.2 it will describe a general architecture for online massive real-time multiplayer games, named *MMORPGs*, or *Massive Multiplayer Online Role-Playing Games* [122, pp. 1–2]. It will be illustrated the implementation and the behavior of the proposed gaming architecture, together with the fairness problems that may be obtained. In Section 6.3 will be sketched the model of the proposed architecture, and defined the set of parameters that will be adopted in the model to characterize the gaming and the network issues used in the simulations. In Section 6.4 it will discussed the implementation of the *client migration*, devoted to the improvement of the game fairness issues. In Section 6.5 it will present the experimental results of the simulation tests. Different imple-

mentations of the migration scheme and heuristics will be discussed and evaluated. Conclusions and future works will conclude the chapter.

6.2 Online gaming architectures

Big multiplayer gaming systems are usually based on a pool of server geographically distributed across one or more countries [74]. These servers share the knowledge about the 'simulated world" where the characters move and act. When a user wants to play the game in multiplayer mode, he/she is required to connect to one of the available servers, where the game activity is currently on, then he/she starts playing. The choice of the server is usually made by following two possible implementations: *i)* a round-robin DNS entry or *ii)* the "nearest server" selection. The concept of "near" server is usually intended as the "geographically nearest" to the client. Both of these solutions are not guaranteed to provide the best performances. Another solution is to connect to the "topologically nearest" server, i.e. the server reachable by traversing the minimum number of intermediate routers within the network structure. Additional advantage could be given by taking into account the congestion along the path to connect to the server.

The goal is to analyze how the communication between a client and one of the servers could be improved without touching the network communication protocols, and by creating an additional, external layer with respect to the gaming infrastructure [103]. Specifically, the class of games resulting interesting contains the *Massive Multiplayer Online Role-Playing Games* (*MMORPGs*) [122, pp. 1–2]. This class of games is becoming more and more popular on the Internet, and includes multiplayer games that simulate a "virtual world". A great number of characters, either under human or computer control, live and act in the virtual world, performing some cooperative or competitive tasks. MMORPGs are the computer-based counterpart of the well-known RPGs (Role Playing Games) [129]. Each new player creates its own *avatar* with a set of characteristics (e.g. strength, intelligence, initial resources) and starts playing the avatar life in the simulated world, by interacting with other

players in order to complete tasks, to compete with enemies, to collect shared and available resources, and to improve the characteristics and skills of his avatar.

MMORPGs may involve a great number of concurrent players. While in other games, like strategic and first-person-shooters [69, 127], the maximum number of interacting players seldom exceeds a dozen [73], MMORPG servers usually have to manage hundreds, or even thousands of avatars co-existing in the same virtual world [68].

6.2.1 MMORPGs gaming architecture

The typical MMORPGs gaming structure is a simple client-server architecture: each player runs a client, which connects to one out of a pool of many servers. The client and the server start to exchange gaming data as long as the user plays, then the client disconnects. Basically, the gaming protocol governing the system evolution is a simple request/reply scheme: clients send commands to one server, and servers synchronize and send "replies" to the clients. A client/server connection starts when a client sends a login request to one of the servers. In the performed simulations, it is implemented a round-robin DNS entry selection of the server to connect to. When a server receives a login request, it performs a service admission control, to admit a limited maximum number of clients, and then it sends a reply, either an *accept* or a *reject* message to the client. Upon a negative answer, the client will retry with another server in round robin fashion until a server accepts it. After a successful "login" the client starts sending play requests that will be received by the server, enqueued and processed on the next timestep boundary. After having processed a request, the server replies to the client by sending him the updated information about the "world" evolution. This communication and synchronization process goes on until the client stops playing. When the client stops playing, it sends a logout request to the server, it waits for an acknowledgment, and then it quits the game. To maintain full synchronization for inactive players, servers send proactively updates to every client in every timestep. The implemented server synchronization protocol

is trivial: each server periodically sends the system status updates to every other server.

The assumptions include that the servers are connected to each other by high speed links. Every server maintains a replica of the simulated world where the locally managed subset of avatars evolves by executing the player's commands. Every player continuously sends unicast messages to its server, while servers send update information to every other server in order to manage the time synchronization and updates of the virtual environment. This is an inefficient way [59] to handle synchronization, and state sharing, in particular when the number of servers is high. The number of messages among servers grows as $\Theta(n^2)$ where n is the number of servers. These system (and modeling) assumptions for this work were selected in order to consider the most general scenario, and a "worst case" implementation that would be highly affected by the distributed communication bottlenecks. It is worth noting that this choice about the server-to-server communication would not affect positively the results of the proposed migration mechanism. Also, the model do not depend on a specific network topology.

6.2.2 Time management and fairness

Let's define as Virtual Time (VT) the time in the virtual world, as Simulation Time (ST) the time concept considered for synchronization issues in the distributed simulation, and as Wall Clock Time (WCT) the time we are living [78, pp. 27–30]. The VT is defined by discrete *steps* of fixed duration (typically in the order of tenths of a WCT second). During each VT step a client can perform at most one *move*, that is, a single command can be sent from the client to the server. With this system (and modeling) assumption, can be controlled the maximum speed of every player under a common upper bounded speed limit. Every command and update received during step $t-1$ is enqueued on the server managing the avatar, and it will be processed at the beginning of the step t in FIFO order, by causing an update of the virtual environment. The timestepped approach for time management introduces a controlled upper bounded delay for the execution of commands. This

can be exploited to introduce a positive fairness principle for clients' commands. The fairness can be improved because fast clients can be limited to perform only one move per timestep, and the timestep size can be tuned such that the same opportunity can be given, on the average, to the majority of slow clients. In this way, all the clients achieve more chances to fairly compete under the remote control by their respective players. Obviously, the timestep duration is a tuning parameter and should not be excessive, in order to maintain a smoothed game evolution. In addition, a MMORPG presents the same issues and the same concept of a real-time simulation. Like in real time simulations, under the interaction viewpoint, the speed gap between the VT of the avatars and the WCT of the players must be negligible.

6.3 The gaming architecture model

In this section it will be described a simple and generic model for the architecture of a massive online multiplayer gaming architecture. The target of this modeling and simulation investigation will be a qualitative comparison between a "classic" gaming and network infrastructure scenario and the same scenario with the effects of the migration mechanism. The definition of an accurate network model or a specific network infrastructure, is out of scope for this goal, because it would be difficult to design and would result in less general results [77]. For the same reasons, I do not aim to define an accurate model of a specific game, and its architecture, but is more interesting to characterize a class of online interactive games in general. On the other hand, I would like to capture the network and gaming infrastructure dynamics and characteristics in our model, in order to obtain significant results. To this end, it was performed an accurate selection of modeling parameters and their respective values and distributions.

6.3.1 The model parameters

The communication protocol at the basis of the proposed architecture has been developed after the accurate analysis of network traffic traces obtained by different real

games [40, 43, 58, 90, 110]. Since most of these games are commercial and proprietary it was impossible to directly examine the protocol details. The only possibility was to capture and analyze the real network traffic generated, to infer the parameters' values to be adopted for the communication protocols' modeling[2]. It must be cleared that the traffic model obtained is characterizing the considered gaming application in isolation. Additional network traffic generated by concurrent applications in real systems would increase the network congestion and the communication bottlenecks. This fact is not critical for this analysis and results, since the interest in qualitative comparison of alternative solutions, under the same assumptions.

6.3.2 The simulated network model

The focus of the network modeling is to determine transmission times for the packets, over generalized network infrastructures (inspired by the Internet).

As in real scenarios, it was modeled a packet based network communication based on a standard TCP/IP Internet protocol architecture. On the other hand, it was not included a detailed management of transmission failures and packet retransmissions. The network is assumed to be reliable, since the scope of this analysis does not include fault-tolerance, routing and link failure issues [51]. It is assumed that communication paths are reliable and the effect of network congestion are modelled as a variable network latency parameter. This latency effect represents the only overhead effect of network congestion included in the model. Packet loss will be modeled as consequence effect of communication delays. By relaxing most of network details, it is assumed that transmission times are mainly determined by *link bandwidth* and *link latency* network parameters [77]. It is assumed that the network is organized in a numbered set of A *areas*, roughly corresponding to big Autonomous Systems, Internet Service Provider networks or Internet carriers (see Figure 6.1).

A parameterized matrix L is defined, such that each pair of network areas, e.g. $(i, j), 0 \leq i, j \leq A$, is assigned a *base latency* $L_{i,j}$. This base latency value pa-

[2]Thanks for their help the system administrators of Midian and Nexus gaming networks' infrastructures [19, 20]

rameterizes a right-shifted lognormal distribution. This distribution is adopted by a pseudo-random generator, to obtain synthetic samples of the latency values for simulated packets traveling from area i to area j and vice versa. The matrix is configured with low base latency values when $i = j$ (i.e. when the two hosts belong to the same area). The right-shifted distribution offset is motivated by the characteristics of the link technology, introducing a 'lower bound" for their respective latencies. Dynamic congestion can be modeled at runtime during the simulation by varying the base latencies values in the matrix L. Each client and server host is assigned a *connection class* that identifies the type of connection the host adopts (i.e. analog modem or DSL for clients, various kinds of ATM links for servers). Each connection class corresponds to a nominal *bandwidth* value, affecting the speed of packets sent and received by that host h. This parameter is also used to define a latency multiplier factor K_h that will be used to determine the network latencies. Both servers and clients are distributed in uniform way in the network areas. When one host (h_i) belonging to area i sends a packet to another host (h_j) belonging to area j, we assume that this packet will follow a path whose end-to-end latency can be modeled as follows. The end-to-end latency is calculated as a summation of "intra-area" and "extra-area" latencies and transfer time. Transfer time is defined as the packet size divided by the link bandwidth of the slowest link in the path, with no variability introduced by Medium Access Control and Data Link policies. As an example, a packet sent with an analog modem comes out from the modem at 33 Kbps, no matter how much fast it will travel in the rest of the network.

If the hosts h_{i1} and h_{i2} belong to the same area i, we call the end-to-end latency as "intra-area latency". It results that "intra-area base latencies" can be found on the diagonal of the matrix. In this scenario, the end-to-end latency is determined as a composition of two expressions:

$$Latency(h_{i1}, K_{h_{i1}}, h_{i2}, K_{h_{i2}}) \quad = \quad K_{h_{i1}} * L_{i,i} + K_{h_{i2}} * L_{i,i} \tag{6.1}$$

where $K_{h_{i1}} * L_{i,i}$ is the initial "intra-area latency" of host h_{i1}, and $K_{h_{i2}} * L_{i,i}$ is the final "intra-area latency" of the destination host h_{i2}, $\forall h_{i1}, h_{i2} \in i$.

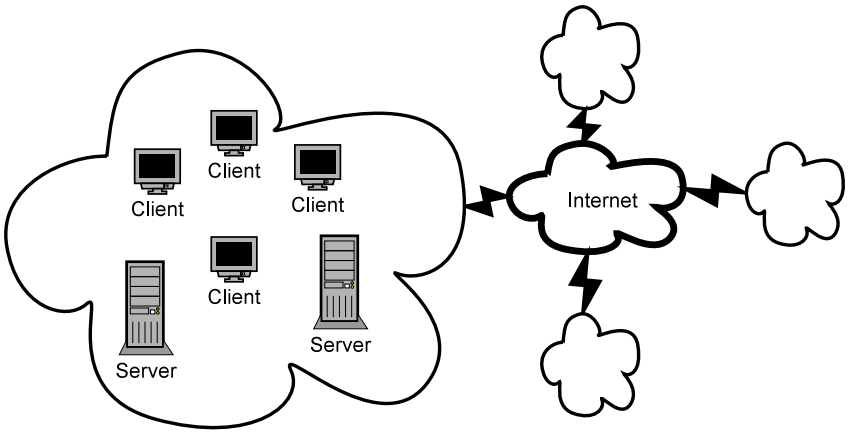

Figure 6.1: An example of the network areas

If h_i and h_j belong to different areas, the overall latency is determined as a composition of three expressions:

$$Latency(h_i, K_{h_i}, h_j, K_{h_j}) = K_{h_i} * L_{i,i} + L_{i,j} + K_{h_j} * L_{j,j}, \qquad (i \neq j) \qquad (6.2)$$

where $K_{h_i} * L_{i,i}$ is the initial "intra-area latency" of area i, L_{ij} is the intermediate "extra-area latency" which is the latency of the network connecting the originating and destination areas, and $K_{h_j} * L_{j,j}$ is the final "intra-area latency" of the destination area j.

The choice of the model parameter values (distribution parameters [41, 104], connection classes characteristics [112], percentage of connection classes) has been made *i)* with direct tests and traffic analysis of real network connections [90, 58], *ii)* by gathering data from some Italian ISPs, carriers and important web sites, and *iii)* by collecting information from other papers and articles.

In Figures 6.2 and 6.3 are shown the experimental ping-time (latency) results obtained when investigating the latency distribution inside the GARR network, and between the GARR network and the Teleglobe network (belonging to two different autonomous systems). This shows that the typical latency distributions for ping times, both inside the same area (Figure 6.2) and across two or more areas (Figure 6.3), could be approximated by a right-shifted lognormal distribution [102, 96].

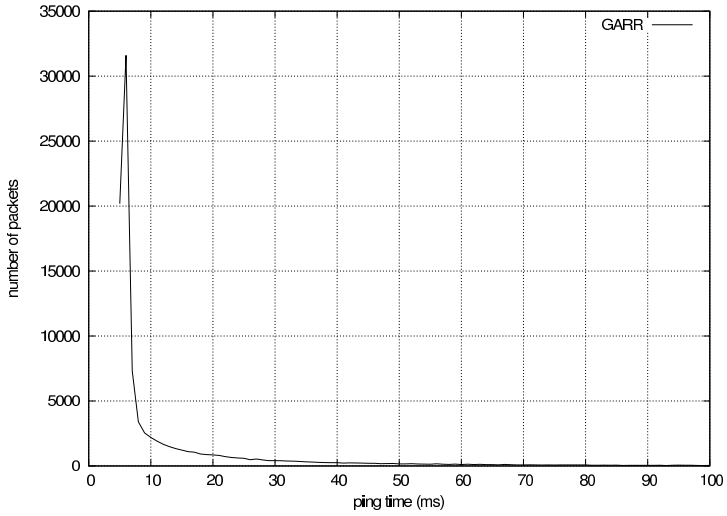

Figure 6.2: Latency inside the GARR Network [8]

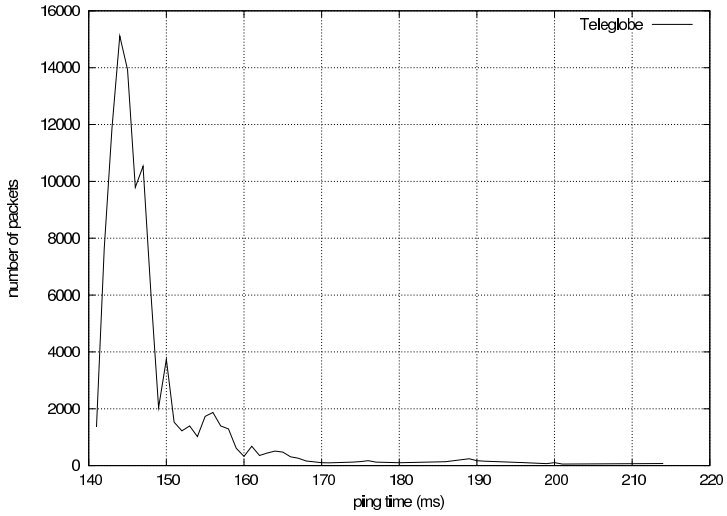

Figure 6.3: Latency between the GARR Network and the Teleglobe Network

Name	Bandwidth (KB/sec)	Presence
DS3	44,736	30%
STS-1	51,840	10%
STS-2	103,680	20%
STS-3	155,520	10%
STS-6	311,040	10%
STS-9	466,560	10%
STS-24	1,244,160	10%

Table 6.1: Model parameter distribution of carriers and big ISPs

The distribution of the various connection classes for carriers and big ISPs has been determined for our model, thanks to information from the Italian Internet Exchange [105]. The resulting information about link-types, link-bandwidths and percentage of "presence" values are reported in table 6.1.

The distribution of the various connection classes for gaming clients has been determined for our model, thanks to information from the Midian gaming network [19]. The resulting information about line-types, upstream and downstream line-bandwidths, connection class latency multipliers (K_h) and the percentage of "presence" values are reported in table 6.2.

To complete the network model parameterization it was analyzed the traffic generated by real games with the aim of discovering information about the packets size distribution and packets' correlation. The proprietary protocols and packet formats used by gaming platforms hidden all details, hence the observation was based on network monitoring. It was discovered that while clients send out few different types of packets, a server generates variable sized packets. Anyway, the typical packet size is 42 bytes. Hence, it was considered as the standard play request/reply packet size, and we used different sizes for other types of event messages.

	Up (KB/sec)	Down (KB/sec)	Latency multip.	Presence
POTS 36.6K	27	27	5.0	4%
POTS 56K	40	25	4.0	24%
ISDN 64K	64	64	3.0	20%
ISDN 128K	128	128	3.0	6%
ADSL 256K	128	256	2.0	20%
ADSL 640K	128	640	2.0	14%
Optical fiber	10000	10000	2.0	12%

Table 6.2: Model parameters distribution of gaming clients

6.3.3 The server model

While it was possible to find good data and references about traffic and network characterization of modeling parameters [40, 72, 73], some difficulties was experienced in finding "real world" values for the server computation model parameters (e.g. CPU times). Hence, to characterize the CPU computation time distributions of the servers was executed external observations, and was measured the time passed from the arrival of a request to the server and the departure of the reply.

6.4 The migration mechanism

In classical gaming implementations, each client chooses a server at the beginning of its playing, and keeps using it until the end of the playing session. Currently, there is no information to assist the client in making a "good choice" at the beginning of the game session. For this reason we suggest the introduction of a *client migration mechanism* in our architecture. The migration mechanism would allow a client to disconnect from a server, and to connect to another server, while continuing to play in user's transparent way. The migration mechanism should be activated when a client "realizes" that the server connection is slow, or another server is reachable by

means of a more fast connection. In this way a better exploitation of the overlay network of server replicas could be performed. One critical point about the migration mechanism is the design and implementation of the migration heuristics to assist clients in the decision to migrate, and also in the selection of a new destination server. In the following it will be described two approaches that was considered in this analysis.

As said before, the communication between a client and a server is affected by two important factors: bandwidth and latency. From the client viewpoint, the bandwidth factor has less influence over the game performances: a high percentage of the packets are quite small [50] so that they can be sent on a low bandwidth link without saturating it. Anyway, the effect of a low bandwidth link can be mitigated by sending less detailed information to the players on that link. The most critical parameter we are dealing with is the end-to-end (E2E) network latency. The E2E latency is a composition of delays introduced by single network links realizing a path from the client to the server and vice versa. Most of the times, the latency of the first hop (usually from the client to the router of its area boundary) cannot be negotiated or controlled by the client. On the other hand, the client can control the game server to connect to, and this allows the client to have many possible alternative paths with different latency performances to equivalent servers. While the first communication step usually has quite low latency variance, the path between the originating ISP and the remote server can have much less predictable latencies due to external Internet traffic aggregation. Due to the network latency variations, the initial choice of the optimal server could not be valid indefinitely. To maintain the optimality, a runtime adaptive selection and migration to adapt to the network conditions is preferable. The client migration can improve the fairness, and can be applied with a limited amount of work on the existing game code, while other methods would require relevant changes to the protocols [99, 121].

Big software houses that create and distribute online multiplayer games consider the network latency a serious problem, such that they manage and invest in servers and networks to offer their customers good network performances and to be compet-

itive with other game producers. Other gaming networks [7, 21, 28] are independent to software houses that produce games, and appear to be quite open to publish technical details about their network structures and protocols.

To maintain the E2E latency upper bounded, many gaming infrastructures, and specifically MMORPGs infrastructures, consider the network partitioned in different wide *areas*, covering whole countries or group of countries [110]. Unfortunately, to maintain massive and global gaming infrastructures, areas can be wide, including multiple servers that appear with different performances to clients. This work's in favor of adopting a migration algorithm, for the motivation above.

Intuitively, the aim of the migration mechanism is to make the latency curve distribution of the clients as much homogeneous as possible. This means that the game service is fairly available, because the latency of a client is similar to the latency of other clients. A timestepped time management like the one defined in previous section could additionally play in favor of the fairness and latency clustering-effect of clients, by letting "fast" clients to be synchronized to the rhythm sustainable by "slow" clients.

6.4.1 The migration heuristics

When a client determines it should try to move to another server, it has to decide which one is the "best choice" for its new connection. As said before, the link bandwidth is not a critical factor, and cannot be modified, because the slowest network segment usually is the one which connects a user to his ISP.[3]

A good migration heuristic should consider at least two other factors: the *network latency* between the client and the server, and the *server load*. It is assumed that each client performs periodic pings towards every possible candidate server, to obtain dynamic estimates of the network latencies. This method is simple, it gives quite a good approximation of the "real" latency and allows a experimental validation of assumptions. Moreover, the ping rate allows to control the overheads introduced.

[3]For instance, a user connecting with a 56Kbps modem is mainly limited by this bandwidth bottleneck, not by his server's one.

The pitfall of the ping method is that a well connected server could result the target for many clients connections, and this may saturate the server capacity. In the proposed architecture, it is introduced a simple load balancing policy based on service admission principles: a server keeps track of the number of currently connected clients and refuses to accept a new connection if its load becomes too high (causing a global performance loss). In future improvements of this architecture, servers could exchange load indicators, by defining a new dynamic distributed load-balancing technique.

The migration trigger

A very important point is the algorithm that activates a client migration. This work is mainly focused on the simplest of all, named *simple migration*, and in addition, some tests was made on a more complex algorithm that we called *early migration*.

Simple migration

The *simple migration* algorithm is the following: once every a fixed number of seconds the client sends a ping probe (ICMP ECHO REQUESTs) to the servers in the list. If a different server appears to be faster than the current one, then a migration process will be activated. Otherwise, no client migration is performed. This policy is simple and stable, because the frequency of migrations can be tuned by paying some reactivity to network dynamics.

Early migration

The *early migration* evaluates migration opportunities at fixed time intervals, like previous policy. In addition, clients collects runtime statistics on the network round trip time by exploiting commands/replies from/to the server. If the runtime statistics indicate that the current connection is becoming too slow, then the client should try to *anticipate* the migration attempt. Many possible implementations can be obtained by considering different statistics and thresholds.

6.4.2 The migration protocol

The client starts from the first server of the list, and if the migration is considered useful, it sends a packet to its current server by asking him to migrate to a new one. Upon migration request, the current server takes care of contacting the new server. If the new server has sufficient available resources for a new client, the current server sends the client data to the new server. When the new client instance is active on the new server the client is notified and starts sending packets to the new server. Meanwhile, the old server forwards the received packets to the new server and routes back the replies. In this way the migration process is transparent to the client that keeps playing during the migration with no additional delays. If a server sends a negative reply to a migration request, the client asks for the following server in its list, until the end of the list is reached, or the migration is completed. The migration protocol is simple, and can be implemented as a logout/login sequence with some intermediate steps to move the client status data and the packet forwarding during the migration phase. The key assumption is that every server maintains the information of managed clients only, together with the information of the whole virtual environment, so that no environment information needs to be migrated. The migration implementation may be dependent to the game platform. Anyway, our preliminary considerations over generalized gaming architectures shown that dependencies from the core of a gaming system should not require complex adaptation efforts.

6.5 Experimental results

6.5.1 Simulation tool

In order to simulate the proposed migration-based architecture, it was not used any existing simulation tool or runtime, but it was created a dedicated simulator, opportunely optimized for the RAM management, in order to overcome the memory bottleneck in a massive simulation [68]. Another reason for this simulator choice was

the model simplicity and simulator speed. Existing powerful runtime environments (e.g. [60]), would have required a complex management and modeling efforts not motivated by the need for high simulator potential. Given the previously described assumptions, the simulation model is quite simple, it is highly modular, and does not require complex simulation services. We want to simulate MMORPG networks with thousands clients connected at the same time, where each client communicates with at least one server and each server communicates with every other server. We implemented, tested and validated a dedicated event-driven simulator written in C language, with good performances and low memory requirements. The discrete event-based simulation paradigm was preferred to a time stepped simulation [97, pp. 7–9, 93, 94] mainly because the behavior of thousands simulated objects generating few events is more efficient to be simulated via discrete event-based simulation.

6.5.2 Simulation setup

The simulation parameters have been discussed in previous sections. Specifically, it was defined the values of the random distributions parameters and their types, the percentage of clients and servers of each connection class, the size distribution of variable kinds of packets [50], the bandwidth and latency characteristics of common network connection classes, and the CPU time required to manage play requests.

The typical simulated scenario is composed by a network with 20 gaming servers and 2000 concurrent clients. All the simulated scenarios' parameters, like the number of clients and servers, the time each client spends in a connection and the frequency it sends play requests have been chosen in order to reflect a real scenario. A number ranging from 10 to 20 independent runs has been executed for each test in order to achieve a good confidence level [97, pp. 253–259].

6.5.3 Performance results

The main performance index will be referred to as *play time*, which is defined as the time a client has to wait for a reply after it has sent a request. The play time is

expressed in milliseconds (ms) and it is used to compare the *fairness* of a gaming infrastructure under many different scenarios.

The first graph (Figure 6.4) is a comparison between the scenario characterized by the original MMORPGs architecture and the same architecture with the simple migration algorithm (see Section 6.4). The figure shows the distribution of clients (Y axis) with respect to their respective average play time values (on the X axis). The solid line represents the original architecture, while the dashed line illustrates the effect of the "simple migration" mechanism over the same scenario. With the original architecture, the play times appear to be clustered around at least 5 sets of distant local aggregation values. This demonstrates that clients would fall with a certain distribution in groups characterized by different play times performances, resulting in unfair gaming service. The play times appear to be much more concentrated with the simple migration mechanism. There is a small group in the right part of the graph that depends on the clients with very slow connections, and another big group on the left with really good average performances. It is worth noting that, in the migration-enabled system, a timestep around 220 ms would satisfy in fair way a majority of clients, with respect to the classical implementation.

In Figure 6.5 is shown the runtime evolution of a fairness index. The concept of "fairness" in this work may be represented as the *variance* of the average play times of the clients in a given time interval (1 s). If the variance value is low, each player experiences a latency which is similar, on the average, to the others players'. On the other hand, a big variance value means that the play times are much more heterogeneous, hence some players suffer the unfairness of the gaming infrastructure. The original architecture (solid line) leads to higher variance value than the same architecture with the simple migration. After an initial transient phase, the migration-enabled architecture outperforms the classic architecture in terms of fairness. In table 6.3, are calculated the confidence intervals for the estimated play time variance. The narrow interval obtained by the simple migration mechanism confirms the expectations about the generalized and uniform level of fairness obtained by the clients of the gaming architecture. Quite surprisingly, the confidence interval for

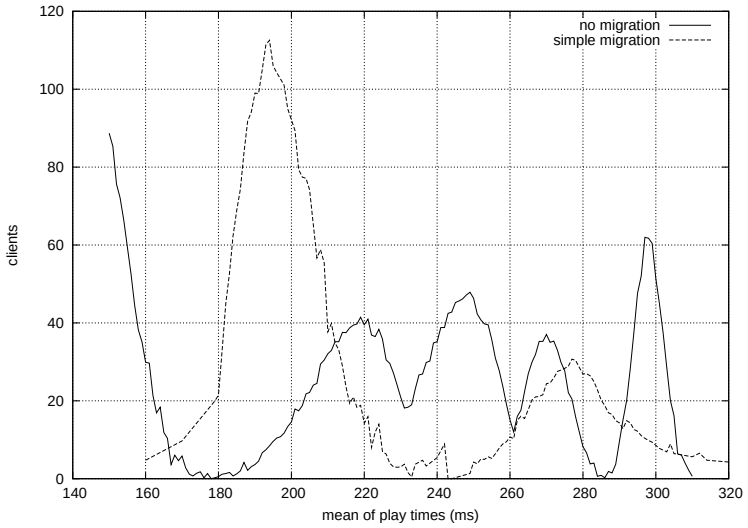

Figure 6.4: Distribution of clients over average play time values: simple migration vs. no migration

the fairness index obtained with the "early migration" heuristic mechanism is larger than the same index in the original architecture with no migration. This was due to the fluctuating behavior of subsets of clients swinging between two servers, by introducing migration overheads and unbalanced loads on the servers.

The analysis on the *average* value of the play times shows that the average time for a reply, i.e. the system interactivity, is favored by the introduction of the simple migration. This can be explained, as we can see from the Figure 6.4, even if some clients are subject to penalty (i.e. the first high spike in the distribution on the left of the solid line is shifted to the right in the dashed one). The penalty for fast clients is quite limited and was considered as a consequence of the architectural design, given the initial assumptions. This fact is not critical, because it does not compromise the interactive potential of fast clients. As a good news, this flattens the differences of play times between fast and slow clients.

In Figure 6.6 it is shown the average play time of messages sent by all the clients in the simulated system during a given time interval. It is clear that at runtime the "simple migration" mechanism gives the better results, with respect to the 'early migration" scheme, which in turn outperforms the classical static (no migration)

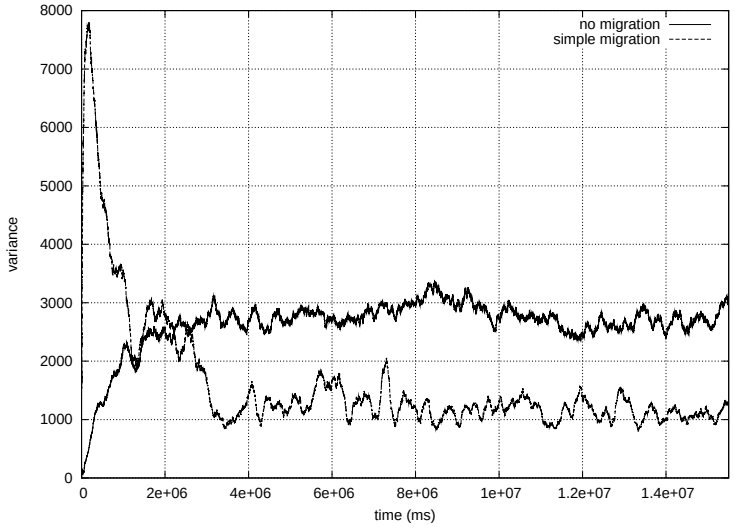

Figure 6.5: Dynamic estimated variance of the play time values (ms)

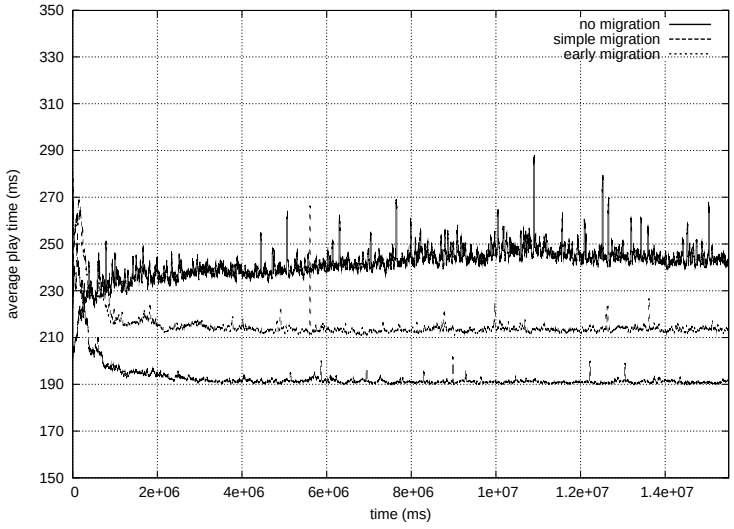

Figure 6.6: Runtime average play time values (ms)

	No migration	Simple migration	Early migration
Run replicas	10	10	10
Run samples	80,221	63,013	62,610
Transient samples	12,000	12,000	12,000
Average Play Time (ms)	2,816.69	1,203.03	1,664.48
Estimation of σ^2	4,766,163.53	965,389.76	4,809,603.78
Play Time Confidence int.	**1,551.23 - 4,082.14**	**633.51 - 1,772.56**	**393.27 - 2,935.69**
Play Time Conf. Interval width	2530.91	1139.05	2542.42

Table 6.3: Confidence intervals of play time variance (90-th percent confidence level)

implementation.

Figure 6.7 shows the same index of Figure 6.6, but this time the X axis is given by the number of moves performed, which is related to the player life duration in the game. The X axis shows the numbered sequence of requests of each client, and the Y axis reports the average play time of the i^{th} request by the set of clients that were involved in the simulation. This performance index needs to introduce some preliminary considerations. First, this index is correct, since the rate of commands sent by slow and fast clients is homogeneous and comparable. Second, each client remains connected for a certain amount of time that has a pre-defined average value [56]. There are many motivations that justify this choice in the model parameterization: mainly, slow clients are usually modem connected, hence they pay for time of connection, and, also, they may experience sudden death of their avatars due to connection delays. The average number of messages sent by slow clients falls around the middle of the X range in the figure, while fast clients perform longer playing activities. The solid line refers to average play times in the architecture with no migration. With this scenario, the average client has a play time of about 250 ms at the beginning of the playing interval, and this index appears to decrease when the slow clients start to abandon the game. After $X = 8000$ moves in the figure,

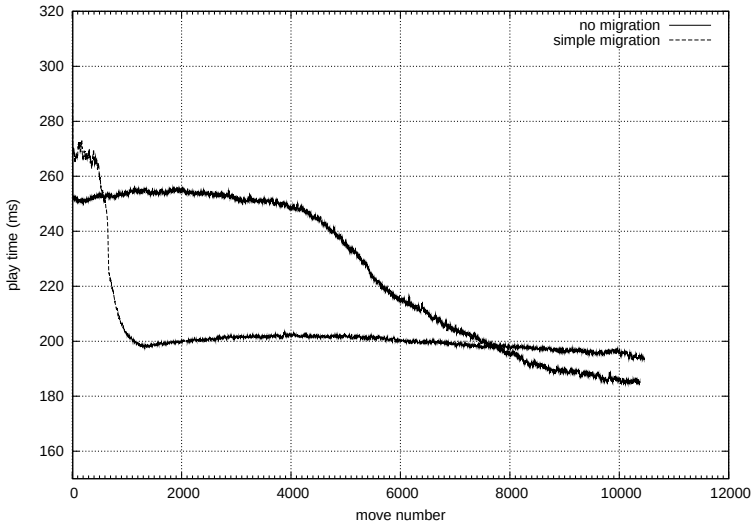

Figure 6.7: Average of play time values move by move

the survived players in the game are mostly fast clients. This is shown in the figure because average play time starts to decrease.

The dashed line shows the average client play times in the "simple migration" scenario. The average play time drops to a low play time after the first migration assessment occurs. The fact that the dashed line converges to a constant value (instead of decreasing like with the solid line) means that every client experiences a comparable play time, which is maintained also when slow clients (e.g. modem enabled) start to leave the game, and this confirms the improvement in fairness.

The fact that the average play time with "no migration" outperforms the play time with "simple migration" after $X = 8000$ moves can be explained because in the "no migration" scenario most of the slow clients already left the system and the reduced number of clients increases the servers and network performances.

The "early migration" heuristic has been evaluated with the same tests performed for the "simple migration" and no migration mechanisms. The results shown that the "early migration" scheme have a worst behavior than the simple migration. A comparison between the two types of migration algorithms is shown in Figures 6.8 and 6.9. The dashed line is for the simple migration, the dotted one is for the early migration. The analysis of simulation traces shown that the early migration

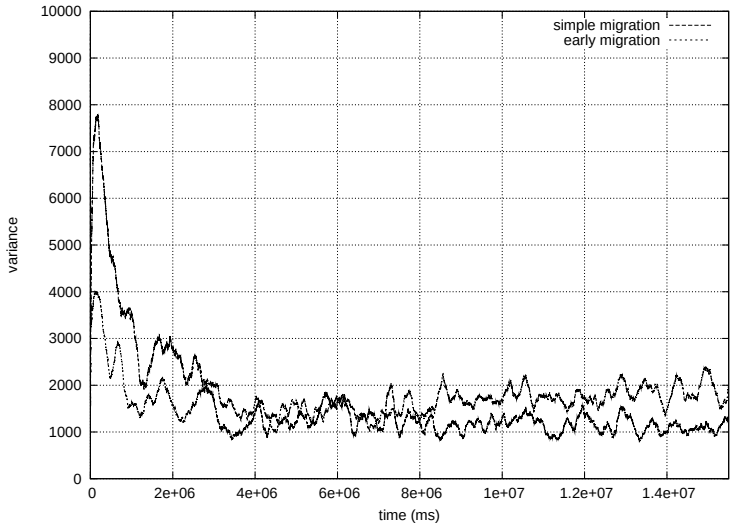

Figure 6.8: Dynamic estimated variance of the play time values (ms), simple vs early migration

scheme has some difficulties when there are two or more servers with a very similar latency. This makes a subset of clients to swing between two servers, by introducing migration overhead and unbalanced loads on the servers.

6.6 Conclusion and future work

In this chapter has been presented an improved architecture model for online multiplayer games, focused on MMORPGs with multiple replicated servers. The proposed mechanism has been simulated over the class of distributed multiplayer and multi-server interactive games, implemented over a general communication network infrastructure. Results show a significant fairness improvement, more homogeneous performances, and the absence of significant overheads, with less or no modifications required to the gaming architecture.

Additional studies will be performed to extend the client migration mechanism: more efficient migration algorithms, new heuristics tuned on new runtime statistics, and the refinement of communication protocols between servers. The creation and design of gaming architectures with "database server" for the storage of player in-

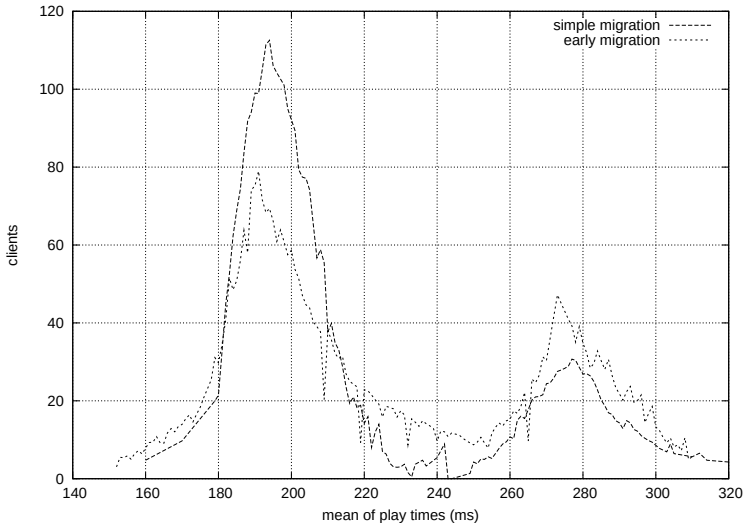

Figure 6.9: Distribution of clients over average play time values, simple vs early migration

formation, and "area servers" for the elaboration of events in different areas of the simulated world [36] will be evaluated. The gaming architecture will be integrated by a pool of "proxies" accepting connections from the clients and forwarding their requests to the right area server [103, 82]. Another research effort will also investigate the possible decoupling and integration of the client migration and the remote and replicated server execution.

As future work we plan to extend the ARTÌS middleware with online gaming capabilities. As stated in the introduction of this chapter we plan to share knowledge between distributed simulation and Internet gaming research fields. Integrating the gaming capabilities in ARTÌS will require a better support for fault tolerance and dynamic client behavior, the implementation of new synchronization algorithms and fulfillment of gaming-related requirements.

Chapter 7

Conclusions

In this thesis has been presented a new middleware for parallel and distributed simulation named Advanced RTI System (ARTÌS) (Chapter 3). In our previously published works [48, 49] it has been demonstrated that many available implementations of the IEEE 1516 standard are unfitted to simulate massively populated dynamic systems. Starting from this analysis, and considering the miss of an efficient Open Source [22] RTI, the ARTÌS middleware has been designed and implemented with special attention to performance issues. The communication and synchronization middleware is adaptive and user-transparent about a large set of optimizations required to improve performances. It is able to detect the best fitted communication mechanism available to take advantage of low latency and reliable communications. The ARTÌS middleware is able of adaptively select the best available interaction module with respect the dynamic allocation of Logical Processes (LPs). The runtime validation of our middleware has been done thanks to the simulation of a set of typical wireless networks (ad hoc and sensor). This simulation effort has involved the analysis of different Media Access Control (MAC) protocols and has included speed-up and scalability evaluation. The ARTÌS middleware has been useful in the development of a new energy aware MAC protocol for static sensor networks (Chapter 4).

The previously described approach has proved to be able to increase the simulation efficiency, but in presence of massively populated dynamic scenarios the

overhall communication overhead could still nullify the speed-up improvements obtained by load sharing. Our approach in this case is based on "migration", dynamically adapting the "simulated entities" allocation over the set of Physical Execution Units (PEUs) involved in the simulation it is possible to drastically reduce the communication overhead. The gain is obtained clustering the "local communications" inside the PEU, without involving costly network communications. This approach is supported in ARTÌS thanks to the integration of the GAIA framework (Chapter 4).

In order to further increase the parallel and distributed simulation speed-up and consequently reduce the overhall time required to complete the simulation runs, we have investigated the "concurrent replication" of parallel and distributed simulation (CR-PADS) (Chapter 5). Executing a set of concurrent simulation runs can bring to better CPU and network -utilization with respect of a "legacy" sequential approach.

Finally, inspired by the parallel and distributed simulation techniques we have designed and simulated a new migration-enabled middleware to support massively populated Internet Games (Chapter 6). In this case the migration is applied to the client-server matching, thanks to this approach the new gaming middleware would be able to reduce communication latency and to significantly improve the game fairness.

In the "conclusions and future work" of each chapter are indicated our planned research directions for the described sub-topics. Generally speaking, the ARTÌS middleware is still far from completeness, a lot of new features are planned or waiting to be integrated. We plan to merge together our "migration based" and "concurrent replication" approaches to obtain an user-transparent middleware characterized by good performance and scalability. Other efforts will be about the tuning and development of better heuristics for both migration and load balancing. In my opinion the developed architecture could be really useful also to address fault-tolerance requiremens and to support the simulation cloning. Until now our synchronization approach has been based on pessimistic algorithms, it could be interesting to evaluate our migration and concurrent replication infrastructure in presence of optimistic

synchronization (i.e. Time Warp). A significant amount of work has been done in the Data Distribution Management (DDM) field and still waits to be accorded to our approach and integrated in our middleware.

Chapter 8

Acknowledgements

It is time to thank people (and not only people) who influenced, in a way or in another, the writing of this thesis and who had an impact on these years of work and research. Thanks to Prof. Azzedine Boukerche who referred this thesis in a very short timeframe and providing really useful remarks. My decision is to write the following of the acknowledgements in Italian. I'm sorry for the readers that cannot understand it, but Italian is my first language, writing in English I would be unable to express all the needed shades of meaning.

Tra le molte persone da ringrazie le prime sono Daniela e Gianni, erano con me quando io ero molto lontano. Un grazie a Lorenzo Donatiello, il mio tutor. Mi ha introdotto nell'affascinante mondo della simulazione, mi ha dato la possibilità di scoprire il serio gioco della ricerca, mi ha permesso di viaggiare e conoscere. Luciano Bononi mi ha insegnato e spiegato innumerevoli cose, ma spesso ha anche sopportato il mio pessimo carattere.

Senza Michele Bracuto molto di quello che ho cercato di raccontare nelle pagine di questa tesi semplicemente non sarebbe stato possibile. Nuovamente grazie all'amica Penelope, sapere che c'è è sempre un sollievo. Roberto ha spesso avuto voglia di ascoltare i miei deliri.

Un grazie anche a molti dei miei colleghi dell'Underground: senza di voi le mie giornate sarebbero state terribilmente noiose e tristi. Quindi grazie ad Andrea, Giorgia, Laura, Valentina, Rocco... no, non ho dimenticato Vas, per lui il grazie è

ancora più grande semplicemente perchè è "fantastico".

In qualche modo devo ricordare anche l'"annus terribilis" 2003, se proprio era necessario mi ha cambiato. Un ringraziamento particolare è dovuto anche a M. per la sua "lectio magistralis" di squallore.

Per chiudere degnamente questo sproloquio voglio riportare una frase di Richard W. Hamming: "*There are wavelengths that people cannot see, there are sounds that people cannot hear, and maybe computers have thoughts that people cannot think*" e quindi grazie anche a: gaia, alice, iaia, sofia, ines, griet, luna, caronte, cerbero, chimera, cassandra, caos, zuma e anche un po' dotto.

References

[1] Cineca scientific computing. http://www.cineca.it/HPSystems/ScientificComputing/index.htm.

[2] Dartmouth SSF (DaSSF). http://www.cs.dartmouth.edu/research/DaSSF/.

[3] Defence Modeling and Simulation Office (DMSO). https://www.dmso.mil/public/transition/hla/.

[4] DMSO: Software Distribution Center. https://sdc.dmso.mil.

[5] Extensible RTI. http://www.npsnet.org/~npsnet/xrti/.

[6] FDK- Federated Simulations Development Kit. http://www.cc.gatech.edu/computing/pads/fdk/.

[7] Gamesnet. http://www.gamesnet.it.

[8] GARR - The Italian Academic and Research Network. http://www.garr.it/garr-b-home-engl.shtml.

[9] GTW/TeD/PNNI. http://www.cc.gatech.edu/computing/pads/teddoc.html.

[10] IEEE Std 1516-2000: IEEE standard for modeling and simulation (M&S) high level architecture (HLA) - framework and rules, - federate interface specification, - object model template (OMT) specification, - IEEE recommended

practice for high level architecture (HLA) federation development and execution process (FEDEP).

[11] Internet engineering task force, MANET WG charter.

[12] iSSF Home Page. http://www.crhc.uiuc.edu/~jasonliu/projects/issf/.

[13] JiST / SWANS. http://jist.ece.cornell.edu/.

[14] Magnetar Games Corporation - Chronos. http://www.magnetargames.com/Products/Chronos/.

[15] Maisie Programming Language. http://may.cs.ucla.edu/projects/maisie/.

[16] MÄK Technologies. http://www.mak.com.

[17] MÄK Technologies. http://www.mak.com/s1ss0p0.php.

[18] Microsoft DirectPlay. http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/direct%x9_c/directx/play/dplay.asp.

[19] Midian server for ultima online. http://midian.gamesnet.it.

[20] Nexus ultima online server. http://www.nexus.sm.

[21] Ngi. http://www.ngi.it.

[22] Open Source Initiative (OSI). http://www.opensource.org/.

[23] OpenSkies - Cybernet. http://www.openskies.net/features/features.html.

[24] OPNET Modeler and Wireless Module. http://www.opnet.com/products/modeler/.

[25] Parallel / Distributed ns. http://www.cc.gatech.edu/computing/compass/pdns/.

[26] Pitch AB. `http://www.pitch.se`.

[27] SNT: QualNet. `http://www.qualnet.com`.

[28] Tgmonline. `http://www.tgmonline.it/fragzone/server`.

[29] Virtual Technology Corporation. `http://www.virtc.com`.

[30] PERF project: Performance Evaluation of Complex Systems: Techniques, Methodologies and Tools, Italian MIUR-FIRB, http://www.perf.it, 2002.

[31] Georgia Tech RTI-kit, http://www.cc.gatech.edu/computing/pads/, 2003.

[32] OMNeT++: discrete event simulation environment, 2004.

[33] Open SystemC Initiative, http://www.systemc.org, 2004.

[34] UCB/LNBL/VINT: the ns-2 network simulator, http://www.isi.edu/nsman/ns/, 2004.

[35] PADS: Parallel and Distributed Simulation group, Department of Computer Science, University of Bologna, Italy. `http://pads.cs.unibo.it`, 2005.

[36] J. Aronson. Using groupings for networked gaming. `http://www.gamasutra.com/features/20000621/aronson_pfv.htm`.

[37] M. Azzolini, M. Bracuto, A. Cremonini, G. D'Angelo, and C. Marchetti. A new ecological approach for information technology. In *Nightmare Group Internal Memorandum*.

[38] R.L. Bagrodia and W.T. Liao. Maisie: A language for the design of efficient discrete-event simulations. *IEEE Transactions on Software Emgineering*, 20(4):225–238, 1994.

[39] R.L. Bagrodia and R. Meyer. PARSEC: A parallel simulation environment for complex system. *IEEE Computer*, 31(10):77–85, 1998.

[40] R.A. Bangun and E. Dutkiewicz. An analysis of multi-player network games traffic. In *3rd IEEE Workshop on Multimedia Signal Processing*, pages 3–8, 1999.

[41] J. Bentini. Architettura distribuita scalabile per applicazioni interattive e multiutente su Internet. Master's thesis, Università di Bologna, 2003.

[42] A. Berrached, M. Beheshti, O. Sirisaengtaksin, and A. Korvin. Alternative approaches to multicast group allocation in HLA data distribution. In *Proceedings of the 1998 Spring Simulation*, Jan 2002.

[43] Bioware Corp. Neverwinter night. `http://nwn.bioware.com`.

[44] Blizzard Entertainment. Warcraft 3. `http://www.blizzard.com/war3`.

[45] L. Bononi, M. Bracuto, G. D'Angelo, and L. Donatiello. ARTÌS: a parallel and distributed simulation middleware for performance evaluation. In *Proceedings of the 19-th International Symposium on Computer and Information Sciences (ISCIS 2004)*, 2004.

[46] L. Bononi, M. Bracuto, G. D'Angelo, and L. Donatiello. A new adaptive middleware for parallel and distributed simulation of dynamically interacting systems. In *DS-RT '04: Proceedings of the 8-th IEEE International Symposium on Distributed Simulation and Real Time Applications*, 2004.

[47] L. Bononi, M. Bracuto, G. D'Angelo, and L. Donatiello. Performance analysis of a parallel and distributed simulation framework for large scale wireless systems. In *MSWiM '04: Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 52–61. ACM Press, 2004.

[48] L. Bononi and G. D'Angelo. A novel approach for distributed simulation of wireless mobile systems. In *Proceedings of IFIP-TC6 8-th International Conference on Personal Wireless Communications (PWC'03)*, 2003.

[49] L. Bononi, G. D'Angelo, and L. Donatiello. HLA-based Adaptive Distributed Simulation of Wireless Mobile Systems. In *Proceedings of the seventeenth workshop on Parallel and distributed simulation*. IEEE Computer Society, 2003.

[50] M. Borella. Source models of network game traffic. *Computer Communications*, 23(4):403–410, 2000.

[51] M. Borella, D. Swider, S. Uludag, and G. Brewster. Internet packet loss: Measurement and implications for end-to-end QoS. In *Proc. of the Intl. Conf. on Parallel Processing Workshops*, 1998.

[52] A. Boukerche and S.K. Das. Dynamic load balancing strategies for conservative parallel simulation. In *Proc. of 11-th Workshop on Parallel and Distributed Simulation (PADS'97)*, pages 20–28, June 1997.

[53] A. Boukerche, S.K. Das, and A. Fabbri. SWiMNet: a scalable parallel simulation testbed for wireless and mobile networks. *Wirel. Netw.*, 7(5):467–486, 2001.

[54] A. Boukerche and A. Fabbri. Partitioning parallel simulation of wireless networks. In *Proceedings of the 32nd conference on Winter simulation*, pages 1449–1457. Society for Computer Simulation International, 2000.

[55] A. Boukerche and A. Roy. Dynamic grid-based approach to data distribution management. *J. Parallel Distrib. Comput.*, 62(3):366–392, 2002.

[56] F. Chang and W. Feng. Modeling player session times of on-line games. In *Proc. of the 2nd workshop on Network and system support for games*, pages 23–26. ACM Press, 2003.

[57] D. Chen, S.J. Turner, B.P. Gan, et al. Alternative solutions for distributed simulation cloning. *Simulation*, 79(5–6):299–315, 2003.

[58] G. Combs et al. Ethereal: A network protocol analyzer. `http://www.ethereal.com`.

[59] E. Cronin, B. Filstrup, and A. Kurc. A distributed multiplayer game server system. `http://citeseer.nj.nec.com/cronin01distributed.html`.

[60] J.S. Dahmann, R. Fujimoto, and R.M. Weatherly. The Department of Defense High Level Architecture. In *Winter Simulation Conference*, pages 142–149, 1997.

[61] J.S. Dahmann, R. Fujimoto, and R.M. Weatherly. The DoD High Level Architecture: an update. In *Winter Simulation Conference*, pages 797–804, 1998.

[62] S.R. Das. Adaptive protocols for parallel discrete event simulation. In *WSC '96: Proceedings of the 28th conference on Winter simulation*, pages 186–193. ACM Press, 1996.

[63] W.J. Davis and G.L. Moeller. The high level architecture: is there a better way? In *Winter Simulation Conference*, pages 1595–1601, 1999.

[64] E. de Souza Mota, K. Pawlikowski, and A. Wolisz. A perspective of batching methods in simulation environment of mulitple replications in parallel. In *Proc. Winter Simulation Conference WSC'2000*, pages 761–766. IEEE Press, 2000.

[65] E. Deelman and B.K. Szymanski. Dynamic load balancing in parallel discrete event simulation for spatially explicit problems. In *Proceedings of the twelfth workshop on Parallel and distributed simulation*, pages 46–53. IEEE Computer Society, 1998.

[66] T.J. Delve and N.J. Smith. Use of dassf in a scalable multiprocessor wireless simulation architecture. In *WSC '01: Proceedings of the 33nd conference on Winter simulation*, pages 1321–1329. IEEE Computer Society, 2001.

[67] K. El-Khatib and C. Tropper. On metrics for the dynamic load balancing of optimistic simulations. In *HICSS '99: Proceedings of the Thirty-second Annual Hawaii International Conference on System Sciences-Volume 8*, page 8051. IEEE Computer Society, 1999.

[68] H. Engum, J.V. Iversen, and O. Rein. Zereal: A semi-realistic simulator of massively multiplayer online games. `http://citeseer.nj.nec.com/555870.html`.

[69] Epic Games, Inc. Unreal Tournament. `http://www.unrealtournament.com`, 2004.

[70] G. Ewing, K. Pawlikowski, and D. McNickle. Akaroa2: Exploiting network computing by distributing stochastic simulation, 1999.

[71] M. Ewing. The economic effects of reusability on distributed simulations. In *WSC '01: Proceedings of the 33nd conference on Winter simulation*, pages 812–817. IEEE Computer Society, 2001.

[72] J. Farber. Network game traffic modelling. In *Proc. of the 1st workshop on Network and system support for games*, pages 53–57, New York, NY, USA, 2002. ACM Press.

[73] W Feng, F. Chang, W. Feng, and J. Walpole. Provisioning on-line games: a traffic analysis of a busy counter-strike server. In *Proc. of the second ACM SIGCOMM Workshop on Internet measurment*, pages 151–156. ACM Press, 2002.

[74] W. Feng and W. Feng. On the geographic distribution of on-line game servers and players. In *Proc. of the 2nd workshop on Network and system support for games*, pages 173–179. ACM Press, 2003.

[75] A. Ferscha. *Parallel and Distributed Simulation of Discrete Event Systems, Handbook of Parallel and Distributed Computing.* McGraw-Hill, 1995.

[76] F.H.P. Fitzek, E. Mota, et al. An efficient approach for speeding up simulation of wireless networks. In *WMSC'2000. Proceedings. of the Western Multiconference. on Computer Simulation*, 2000.

[77] S. Ford and V. Paxson. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking (TON) archive*, 9(4):392–403, August 2001.

[78] R.M. Fujimoto. *Parallel and Distributed Simulation Systems*. John Wiley & Sons, Inc., first edition, 2000.

[79] B.P. Gan, Y.H Low, S. Jain, et al. Load balancing for conservative simulation on shared memory multiprocessor systems. In *Proceedings of PADS 2000*, pages 139–146. IEEE Computer Society, 2000.

[80] L. Gautier and C. Diot. Design and evaluation of MiMaze, a multi-player game on the Internet. In *Intl. Conf. on Multimedia Computing and Systems*, pages 233–236, 1998.

[81] Gravity Corporation. Ragnarok online. `http://iro.ragnarokonline.com`.

[82] C. Griwodz. State replication for multiplayer games. In *Proc. of the 1st workshop on Network and system support for games*, pages 29–35. ACM Press, 2002.

[83] T. Henderson. Latency and user behaviour on a multiplayer games server. In *Proc. of the 3rd Intl. Workshop on Networked Group Communication (NGC)*, pages 1–13, November 2001.

[84] P. Huang, D. Estrin, and J. Heidemann. Enabling large-scale simulation: Selective abstraction approach to the study of multicast protocol. In *Proceedings of the 6th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, page 241. IEEE Computer Society, 1998.

[85] M. Hybinette and R.M. Fujimoto. Cloning: a novel method for interactive parallel simulation. In *WSC '97: Proceedings of the 29th conference on Winter simulation*, pages 444–451. ACM Press, 1997.

[86] M. Hybinette and R.M. Fujimoto. Cloning: a novel method for interactive parallel simulation. In *Proceedings of the 29th Conference on Winter Simulation*, pages 444–451. ACM Press, 1997.

[87] M. Hybinette and R.M. Fujimoto. Dynamic virtual logical processes. In *PADS '98: Proceedings of the twelfth workshop on Parallel and distributed simulation*, pages 100–107. IEEE Computer Society, 1998.

[88] M. Hybinette and R.M. Fujimoto. Cloning parallel simulations. *ACM Trans. Model. Comput. Simul.*, 11(4):378–407, 2001.

[89] M. Hybinette and R.M. Fujimoto. Scalability of parallel simulation cloning. In *SS '02: Proceedings of the 35th Annual Simulation Symposium*, page 275. IEEE Computer Society, 2002.

[90] V. Jacobson, C. Leres, S. McCanne, et al. Tcpdump. `http://www.tcpdump.org`.

[91] D.R. Jefferson. Virtual time. *ACM Trans. Program. Lang. Syst.*, 7(3):404–425, 1985.

[92] K.G. Jones and S.R. Das. Parallel execution of a sequential network simulator. In *Proceedings of the 32nd conference on Winter simulation*, pages 418–424. Society for Computer Simulation International, 2000.

[93] O.E. Kelly, J. Lai, N.B. Mandayam, et al. Scalable parallel simulations of wireless networks with WiPPET: modeling of radio propagation, mobility and protocols. *Mob. Netw. Appl.*, 5(3):199–208, 2000.

[94] F. Kuhl, R. Weatherly, and J. Dahmann. *Creating computer simulation systems: An Introduction to the High Level Architecture*. Prentice Hall, 1999.

[95] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7), 1978.

[96] T. Lang, G. Armitage, P. Branch, and H. Choo. A synthetic traffic model for Half-Life. In *Australian Telecommunications Networks & Applications Conference 2003*, Melbourne, Australia, December 2003.

[97] A. Law and W. Kelton. *Simulation Modeling and Analysis*. McGraw Hill International Series, third edition, 2000.

[98] Y. Lin. Parallel independent replicated simulation on a network of workstations. In *PADS '94: Proceedings of the eighth workshop on Parallel and distributed simulation*, pages 73–80. ACM Press, 1994.

[99] Y. Lin, K. Guo, and S. Paul. Sync-MS: Synchronized messaging service for real-time multi-player distributed games. In *Proeedings of 10th IEEE Intl. Conf. on Network Protocols (ICNP)*, 2002.

[100] W.W. Liu, C. Chiang, H. Wu, et al. Parallel simulation environment for mobile wireless networks. In *Proceedings of the 28th conference on Winter simulation*, pages 605–612. ACM Press, 1996.

[101] J. Luthi and S. Grosmann. The resource sharing system: dynamic federate mapping for HLA-based distributed simulation. In *PADS '01: Proceedings of the fifteenth workshop on Parallel and distributed simulation*, pages 91–98. IEEE Computer Society, 2001.

[102] MathWorld. Continuous Statistical Distributions. `http://mathworld.wolfram.com/topics/ContinuousDistributions.html`.

[103] M. Mauve, S. Fischer, and J. Widmer. A generic proxy system for networked computer games. In *Proc. of the 1st workshop on Network and system support for games*, pages 25–28. ACM Press, 2002.

[104] J.C. McEachen. Traffic characteristics of an Internet–based multiplayer online game. In *4th Intl. Conf. on Information, Communications & Signal Processing*, 2003.

[105] Milan Internet eXchange. Charter members. `http://www.mix-it.net/elenco_afferenti.html?lang=it&order=banda`.

[106] M. Myjak, S. Sharp, et al. Implementing object transfer in HLA. In *SIW '99: Proceedings of the 5-th Simulazion Interoperability Workshop*, 1999.

[107] V. Naoumov and T. Gross. Simulation of large ad hoc networks. In *Proceedings of the 6th international workshop on Modeling analysis and simulation of wireless and mobile systems*, pages 50–57. ACM Press, 2003.

[108] D.M. Nicol and R.M. Fujimoto. Parallel simulation today. *Annals of Operations Research*, (53):249–285, 1994.

[109] M. Oliveira and T. Henderson. What online gamers really think of the Internet? In *Proc. of the 2nd workshop on Network and system support for games*, pages 185–193, New York, NY, USA, 2003. ACM Press.

[110] Origin. Ultima online. `http://www.ultimaonline.com`.

[111] J. Panchal, O. Kelly, J. Lai, et al. Parallel simulations of wireless networks with TED: radio propagation, mobility and protocols. *SIGMETRICS Perform. Eval. Rev.*, 25(4):30–39, 1998.

[112] L. Peterson and B. Davie. *Computer Network — A Systems Approach*. Morgan Kaufmann, second edition, 2000.

[113] D.M. Rao and P.A. Wilsey. An object-oriented framework for parallel simulation of ultra-large communication networks. In *Proceedings of the Third International Symposium on Computing in Object-Oriented Parallel Environments*, pages 37–48. Springer-Verlag, 1999.

[114] D.M. Rao and P.A. Wilsey. Parallel co-simulation of conventional and active networks. In *MASCOTS*, pages 291–298, 2000.

[115] D.M. Rao and P.A. Wilsey. Parallel co-simulation of conventional and active networks. In *Proceedings of the 8th International Symposium on Modeling,*

Analysis and Simulation of Computer and Telecommunication Systems, page 291. IEEE Computer Society, 2000.

[116] D.M. Rao and P.A. Wilsey. An ultra-large scale simulation framework. *Journal of Parallel and Distributed Computing*, 10(1):18 – 38, 2000.

[117] G.F. Riley and M. Ammar. Simulating large networks: How big is big enough? In *Proceedings of First International Conference on Grand Challenges for Modeling and Simulation*, Jan 2002.

[118] G.F. Riley, R.M. Fujimoto, and M.H. Ammar. A generic framework for parallelization of network simulations. In *Proc. of 7th Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecomm. Systems*, 1999.

[119] T. Schulze, S. Strassburger, and U. Klein. HLA-federate reproduction procedures in public transportation federations. In *Proceedings of the 2000 Summer Computer Simulation Conference*, 2000.

[120] J. Short, R.L. Bagrodia, and L. Kleinrock. Mobile wireless network system simulation. *Wirel. Netw.*, 1(4):451–467, 1995.

[121] S.K. Singhal and D.R. Cheriton. Using a position history-based protocol for distributed object visualization. Technical Report CS-TR-94-1505, 1994.

[122] J. Smed, T. Kaukoranta, and H. Hakonen. Aspects of networking in multiplayer computer games. In W.H. Man L.W. Sing and W. Wai, editors, *Proc. of Intl. Conf. on Application and Development of Computer Games in the 21st Century*, pages 74–81, Hong Kong SAR, China, Nov 2001.

[123] T. Smed, J. Kaukoranta and H. Hakonen. A review on networking and multiplayer computer games. Technical Report 454, Turku Centre for Computer Science, April 2002.

[124] T.K. Som and R.G. Sargent. Model structure and load balancing in optimistic parallel discrete event simulation. In *Proceedings of the fourteenth workshop on*

Parallel and distributed simulation, pages 147–154. IEEE Computer Society, 2000.

[125] K. Tang, M. Correa, and M. Gerla. Effects of Ad Hoc MAC layer medium access mechanisms under TCP. *MONET*, 6(4):317–329, 2001.

[126] P. Unold, C.L. Gregersen, A.O. Kjeldbjerg, et al. Freeciv. `http://www.freeciv.org`.

[127] Valve Corporation. Counter–Strike. `http://www.counter-strike.net`.

[128] V. Vee and W. Hsu. Locality-preserving load-balancing mechanisms for synchronous simulations on shared-memory multiprocessors. In *Proceedings of the fourteenth workshop on Parallel and distributed simulation*, pages 131–138, 2000.

[129] Wizards of the Coast, Inc. Dungeons & Dragons. `http://www.wizards.com/dnd`.

[130] X. Zeng, R.L. Bagrodia, and M. Gerla. GloMoSim: A library for parallel simulation of large-scale wireless networks. In *proc. PADS'98*, 1998.

Index