Scalable and Efficient Parallel and Distributed Simulation of Complex, Dynamic and Mobile Systems

Luciano Bononi, Michele Bracuto, Gabriele D'Angelo, Lorenzo Donatiello
Dipartimento di Scienze dell'Informazione, Università degli Studi di Bologna,
Mura Anteo Zamboni 7, 40126, Bologna, Italy
{bononi,bracuto, gdangelo, donat}@cs.unibo.it

Abstract

In this work we illustrate the design and implementation guidelines of a recently developed middleware defined to support the parallel and distributed simulation of large scale, complex and dynamically interacting system models. The distributed simulation of complex system models, may suffer the communication and synchronization required to maintain the causality constraints between distributed model components. We designed and implemented the ARTÌS middleware as a new framework by incorporating a set of features that allow adaptive optimization by exploiting many complex and dynamic model and distributed simulation characteristics. As an example, a dynamic migration mechanism for the run-time adaptive allocation of model entities has been designed and exploited for dynamic load and communication balancing. Optimizations have been introduced to obtain the maximum advantage from heterogeneous and asymmetric communication systems, from shared memory to LAN and Internet communication. Other optimizations have been introduced by the exploitation of concurrent replications of parallel and distributed simulations, in order to increase the resources utilization and to maximize the speedup of simulation processes. Solutions have been designed, implemented and tuned to obtain a significant reduction in the communication and synchronization overheads between the physical execution units, and an increased model scalability and simulation speedup, even in worst-case modeling assumptions and simulation scenarios.

1. Introduction

The simulation attempts to represent certain features of the behavior of a physical or abstract system by the behavior of another system. In other words, a computer simulation is a program execution that manages and updates a set of model variables. The model is an abstraction of the real system, and it is composed by model entities. Each model entity is defined by the data structures implementing the entity state, and by the executable code that represents the entity-state evolution. In discrete-event simulation, the evolution is event-based, that is, one event is the cause for state changes, occurring at discrete time instants. One event is processed by one execution unit by executing an event handler process. The execution of one event may schedule new events in the future, or cancel the future execution of scheduled events. A simulation process is a process that incarnates the ordered events' execution, and the behaviors of at least one model entity. A synchronization mechanism is needed to order the events' executions, by following their causal order. The causal order can be informally defined as the notion of "happens before" ordering of events. The real system evolution and the interactions between system entities is represented by the model state variables' updates. The causal order of events can be obtained through *i)* a totally ordered event list in monolithic simulators, and *ii)* a distributed synchronization mechanism implemented through message passing communication of event notifications, in parallel and distributed simulators. It results a great importance of the communication efficiency in distributed simulation scenarios.

In recent years, the research for tools and methodologies for modeling and simulation of large-scale and complex systems has obtained a great interest. Examples of models challenging currently available simulation systems and tools range from large-scale wireless systems, like cellular, mobile ad hoc and sensor networks, up to biology-inspired models and molecular systems, elementary particles physics and cosmology systems [26, 24, 27, 28]. The simulation-based investigation of complex systems is widely adopted and it is often preferred, in practice, to the complexity of alternative numerical and analytical modeling and resolution methods [23, 28]. Simulation models currently considered interesting for the analysis may include a potentially huge number of simulated objects. Such simulations may require a relevant computation time to complete (e.g. due to the implementation of complex behaviors, with dense and computation intensive state updates). Detailed and complex simulated objects (model components) may require complex and large data

This work was supported by the Italian Ministry of Education, Universities and Research (MIUR) in the framework of the FIRB-Perf project:"Performance Evaluation of Complex Systems: Techniques, Methodologies and Tools".

structures implementing the model state. For these reasons, large scale and complex simulation models may be 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 [27, 28].

Many practical experiences have demonstrated that the memory bottleneck reduction, the model scalability and the speed-up in the simulation of complex systems, can be achieved by using parallel and distributed models and execution architectures, i.e. a Parallel Discrete Event Simulation (PDES) approach [1, 9, 16, 17, 26, 28]. Parallel and Distributed Simulation (PADS) is the acronym used to refer to execution of concurrent simulation processes over tightly coupled, or loosely coupled computation architectures, respectively. The advantage of PADS is given by the exploitation of aggregate memory and computation resources of the Physical Execution Units (PEUs) architectures. More recently, the distributed simulation community contributed in the definition of IEEE 1516 High Level Architecture: a new standard for distributed modeling and simulation [20]. The new standard defines rules and interfaces allowing for heterogeneous components' interoperability in parallel and distributed simulations. Model components (formally known as federates) are executed as Logical Processes (LPs) over a set of interconnected PEUs. Federates' execution is supported by standard management APIs for the communication and synchronization tasks, implemented by a run-time middleware (RTI). The High Level Architecture (HLA) has currently become a synonymous for the standard rules and services to be considered as the basis for the implementation of distributed simulations, and the Runtime (RTI) simulation kernel [11, 15, 20].

In order to exploit the maximum level of computation parallelism, many research activities dealt with dynamic balancing of LP's executions (both cpu-loads and virtual time-advancing speeds) by trading-off communication, synchronization and speedup, both in optimistic and conservative approaches [8, 12, 14, 29, 30, 31]. Under a conservative synchronization algorithm, a LP executes the next event with timestamp *t* only when it is sure that no event-messages will be notified whose timestamps are lower than *t* (safety condition). In the optimistic approach events can be executed without necessarily evaluate the safety predicate condition. If a violation of the event causality is detected, that is, an out-of-order message is received by an LP, a rollback process is executed to resume last correct state.

The distributed federates interact and synchronize via event-message notifications (basically, message passing communication). Unfortunately, the need for distributed model-components communication and synchronization services may require massive interprocess communication. 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 main bottleneck of the distributed simulation paradigm, and solutions to reduce the cost of communication must be addressed by the research in this field.

Many approaches have been investigated in order to reduce the overhead effects of distributed synchronization and communication in both optimistic and conservative distributed simulations. Solutions have been proposed, based on both model aggregation and on communication filtering, and also by trading off model accuracy and computation load balancing issues, respectively [19, 25]. Basically, the approaches defined in [2, 9, 14, 15, 22, 29, 30] rely on the reduction of communication obtained when the update of an event- or state-information (e.g. event-messages) is not flooded to the whole system, but it is propagated only to the subset of causally dependent components. This is the basis of publishing/subscribing mechanisms for sharing state-information and event-notifications between causally dependent components [11, 15, 26]. In spite of the previously mentioned approaches for communication reduction, the efficient implementation of interprocess communication remains a primary background issue, to contrast the possible communication bottleneck of parallel and distributed simulations.

Recently implemented middleware solutions based on the High Level Architecture (HLA) [11, 20] have shown that the parallel and distributed simulation of massive and complex systems can result in relevant overheads. Overheads are due to the complex and full management of a wide set of run-time services and to the latency due to distributed communication bottlenecks. Specifically, most of the preliminary implementations of the interprocess communication services have been implemented in sub-optimal way, without considering the heterogeneity of the simulation execution platforms [3, 13].

We designed a new, parallel and distributed simulation middleware named Advanced RTI System (ARTÌS). The aim of the ARTÌS middleware is to support parallel and distributed simulations of complex and dynamic systems. ARTÌS design is oriented to support the model components' heterogeneity, distribution and reuse, and to increase the simulation performances, scalability and speedup, in parallel and distributed simulation scenarios [4]. ARTÌS is designed to implement dynamic adaptation of the interprocess communication layer to the heterogeneous communication support offered by possibly different simulation-execution units. Specifically, ARTÌS performs adaptive evaluation of the communication bottlenecks. It supports multiple communication infrastructures and services, from shared memory to MPI, LAN and Internet-based communication [4].

ARTÌS has been integrated with the Generic Adaptive Interaction Architecture (GAIA), to support a simple model components' migration mechanism that can be adapted on the top of HLA-based distributed simulations [3]. The adoption of GAIA enabled a dynamic partition and allocation of interacting model components, allocated over the LPs, and respectively executed over a set of multiple, distributed PEUs. In this way, we obtained an adaptive, tuneable mechanism able to adapt and react to dynamic systems' behavior under the load balancing and communication-reduction viewpoints. In addition, many other solutions and optimizations have been introduced in the ARTÌS framework. As an example, the support for Concurrent Replication of Parallel and Distributed Simulations (CR-PADS) [6] is realized to maximize the speedup and utilization of system resources, when implementing a set of parallel and distributed simulations of complex dynamic system models. More recently, three classes of solutions have been proposed to improve the performance of simulations executed over commodity off-the-shelf computation and communication architectures: multi-threaded software and Hyper-Threading support by the processor architectures, data marshalling solutions for shared-memory and network-based communications [7]. In this paper we sketch the illustration of all the main issues and results achieved in the ARTÌS design and implementation.

The paper structure is the following: in section 2 we outline the state of the art of PADS and existing runtime implementations; in Section 3 we illustrate the key design and implementation issues of ARTÌS; in section 4 we outline solutions for dynamic and complex systems' simulation; in section 5 we illustrate a solution to improve resources' utilization and simulation performances; in section 6 we sketch some optimizations; in section 7 we summarize our conclusions and future work.

2. State of the art

One of the main issues for PADS was the lack of a modeling and simulation interoperability standard. This resulted in many duplicate model implementations, inability to share models and information between heterogeneous simulators and a very low code reuse. In 2000, many years of research work involving both public and private units, led to the IEEE 1516 standard High Level Architecture (HLA) being approved [15, 20]. The HLA standard defines APIs for the communication and synchronization tasks among federates. The distributed simulation is supported by a runtime middleware (RTI), whose implementation techniques are not regulated by the standard. The RTI is mainly responsible for providing a general support for distributed objects' interaction, attributes' ownership and many other optimistic and conservative event-management policies. The IEEE 1516 standard has gained a good popularity but it is still considered too complex for the implementation of distributed simulation middleware solutions and architectures. The existing ones are often too slow in supplying the expected results. Specifically, the IEEE 1516 Standard has been criticized about its structure and its effective ability to manage really complex and dynamic models [13]. By analyzing the existing RTI implementations, to the best of our knowledge, few currently available middleware solutions have been designed with some emphasis on the adaptive exploitation of the communication infrastructure heterogeneity, which may be characterizing the distributed simulation-execution scenario. More specifically, the Georgia-tech RTI-kit [18] implementation has been realized by introducing some optimization in the exploitation of the shared memory execution-system architecture. Many other implementations still rely on UDP or TCP socket-based interprocess communication, even when executed 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 the execution-system architectures. The support for heterogeneous communication services and architectures should be considered as a design principle in the implementation of a distributed simulation 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. More specifically, none of the proposed RTI implementation took under consideration the specific assumptions and characteristics of simulation execution of complex, dynamic and mobile system-models. Our work on the ARTÌS implementation introduced these concepts, to provide a scalable, adaptive, easy to configure and efficient execution and communication support for PADS.

3. The ARTÌS software architecture

The ARTÌS implementation follows a component-based design, that results in an extendible middleware suitable for many applications. Currently, ARTÌS supports both the conservative and optimistic synchronization. The former is implemented with both the time-stepped approach, and the Chandy-Misra-Bryant (CMB) algorithm [10]. The latter relies on a Time Warp algorithm implementation [21]. In our experience in the simulation of complex and dynamic systems, the conservative approach resulted more efficient than the optimistic one. This is due to the highly unpredictable characteristics of dynamic and mobile models of interest. Under the optimistic approach, the simulation has given frequent rollbacks, by increasing the simulation overhead.

In ARTÌS, preliminary design optimizations have been evaluated for synchronization and communication protocols in Local Area Network (LAN) or Shared Memory (SHM) multiprocessor architectures. The communication and synchronization modules should be adaptive and user-transparent about all adaptation and optimizations required to improve performances.

Figure 1 shows the current 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 to select the best interaction module with respect to the dynamic allocation of Logical Processes (LPs) in the execution environment. As an example, the presence of a shared memory for the communication among parallel or distributed Logical Processes (LPs) offers the advantage of low latency, and reliable communication mechanism. The shared memory communication module required accurate design, implementation and testing. The optimal module is highly scalable, easy to manage, and with low latency and overheads. We tested solutions based on semaphores and locks that showed high latency and scaling problems. The number of semaphores that could be instantiated in a system is often limited and statically controlled by the operating system kernel. Busy-waiting solutions drastically reduce the latency, but introduce high CPU computation overheads. In ARTÌS, the current shared memory 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 overheads. Moreover, it resulted in good performance obtained in multi-CPU systems, good scalability and no need for any reconfiguration at the operating system kernel level.

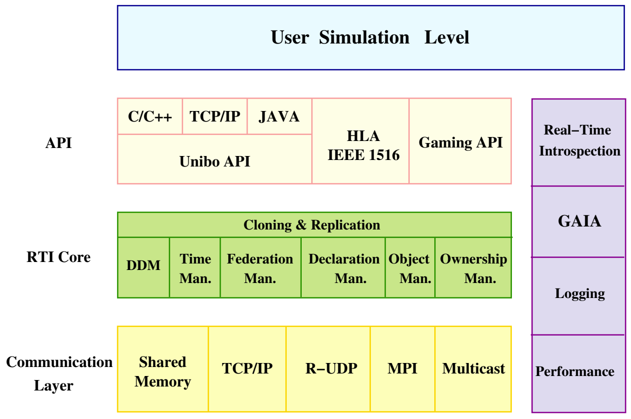

Figure 1. The ARTÌS software architecture

When shared memory is not available between two or more LPs located on different hosts, the other modules support LPs' communication by using either a light Reliable-UDP/IP (R-UDP/IP), or the standard TCP/IP protocol stack. The ARTÌS choice is based on the network characteristics: in a reliable LAN, the R-UDP solution will be preferred for the low overhead. In general Internet–based simulations, the TCP/IP stack will be adopted. Recently, the module support for the Message Passing Interface (MPI) has been integrated in the middleware. The MPI is a communication protocol which represents a "de facto" standard for communication among nodes in the High Performance Computing (HPC) architectures.

On top of the communication modules' layer, the ARTÌS runtime core layer (RTI Core) is composed by a set of management modules directly inspired by a typical HLA-based simulation middleware, and made compliant with the IEEE 1516 Standard.

Many different modules have been implemented: Data Distribution Management (DDM) in charge of managing the dynamic subscription/distribution of event- and data-update messages, the Time Management, the Federation Management, Declaration Management, Object Management and Ownership Management. Most of those modules are in preliminary stage of design and implementation and will be refined in future, by adding more features. Currently, our primary goal was to obtain a easy to use middleware, suitable to run complex and dynamic models, and working as test-bed for new features and experimentations reported in this paper.

The simulation middleware services are exported by ARTÌS to the upper user simulation layer, by implementing a set of modules implementing different APIs. The HLA IEEE 1516 API module is implemented to allow the integration of IEEE 1516 compliant models to the ARTÌS framework. The native set of APIs, called the University of Bologna APIs (Unibo APIs), offers a simpler access to distributed simulation services, with respect to IEEE 1516 Standard APIs. The bindings for the Unibo APIs are currently provided for C/C++/JAVA languages. Thanks to the recently introduced Java bindings, models and simulators can be written in Java. This allows seamless integration of new Java modules with native modules, and integration of Java simulation models. Currently, we are implementing a specific set of APIs designed for ARTÌS support to Internet Gaming applications. We are currently developing a set of simple test games to identify the optimal API design.

Additional orthogonal modules are dedicated to other specific features, like the adaptive runtime management of synchronization and communication overheads. As an example, a real-time introspection mechanism offers an internal representation of the middleware state at runtime. Logging and Performance modules support the user simulation with online traces, statistics and runtime data analysis. The Migration (GAIA) module will be described in the next section, by illustrating how distributed model entities migrations made simulations of dynamic models more efficient.

The runtime has been recently upgraded with simulation cloning and concurrent replication modules. The cloning module allows the dynamic cloning of simulation executions, and it is useful to concurrently analyze "what-if" scenarios. The concurrent replication model can be exploited to increase the simulation performances (basically, to decrease the WCT) and its contribution will be discussed in section 5.

4. Simulation of massive, complex and dynamic systems

We define a dynamic system as a system where the interactions (i.e. the causal effects of events) are dynamically subject to fast changes driven by the system (and model) evolution over the simulated time. Given this general definition, a wireless network can be an example of a highly dynamic system. To realize a correct evolution under the event-causality viewpoint, every model components' interaction should be notified, as an event-message, to all the causally dependent model components. This task is done by the run-time event message distribution mechanism. The intensive communication inherent to complex systems under the distributed simulation approach makes inter-process communication a potential bottleneck for the distributed simulation paradigm. The way inter-process communication can be sustained in distributed systems depends on the PEUs and on the communication support, that is, on the simulation system resources, architectures and characteristics. As previously said, message passing communication can be performed efficiently over shared memory architectures, while it would require medium and high communication latencies over local and wide area network communication services.

It is self evident how 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 highly dynamic systems any optimal static clustering and allocation, based on the current component-interaction scheme at a given point in the simulation, will become immediately suboptimal, due to the dynamics of the model interactions. In presence of a dynamic system, 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. This 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 interaction scheme would be maintained for a significant time (i.e. time-locality). In our approach the optimization is dynamically performed at run-time, by the GAIA module migrating the model entities between LPs [3]. In this way, the modeller is alleviated by the optimization task, and the system converges towards a balanced, tuneable and pseudo-optimal model components' distribution driven by the model interaction scheme.

4.1. Case study: the wireless and mobile system model

In the following, as an example of a dynamically variable system, we focus on a wireless multi-hop Mobile Ad Hoc Network (MANET) [22]. Simulation models for wireless systems incarnate all the assumptions that motivated our design. The number of simulated hosts in our expectations can reach high values, requiring the simulation of massively populated scenarios. Topology changes due to simulated hosts' mobility map on causality effects in the "areas of influence" of each mobile device, resulting in dynamically shaped causality-domains and component interaction schemes. Given two or more neighbor-hosts sharing the wireless medium, the causal effect of signal interference could result in a complex chain of local-state events up to the high protocols' layer. In our approach, we define a model entity as the data structure defined to model a Simulated Mobile Host (SMH). A certain degree of time-locality of local communication can be considered an acceptable assumption in many wireless system models, depending on the communication load and the mobility model assumptions.

A high degree of causality in the simulation of the wireless hosts' communication is driven by the local topology interaction (i.e. transmissions) between neighbour hosts. Under the modeling and simulation viewpoint, wireless systems can be considered highly dynamic systems: if a SMH changes its position, it will eventually interact with a new community of neighbour hosts. The system dynamics can be influenced by motion model and speed, and also by the SMHs density.

Our testbed consists of a distributed discrete event simulation of model components (i.e. logical processes) executed over a set of physical execution units (PEUs), connected by a physical LAN network. Our 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 run-time simulation core (RTI) as a single simulation component. On the other hand, a single LP is implicitly formed by a set of threads. The main thread of each LP manages and updates the state (i.e. local data structures) of a set of Simulated Mobile Hosts (SMHs). A communication between wireless hosts can be modelled as a set of interactions (i.e. message-events) between any couple of adjacent SMHs. Since a wireless communication must be always modelled 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 obtain the advantage of closing the causality effect of modelled communication within the PEU where the interacting LPs (and respective SMHs) are executed. In addition, clustered interacting SMHs would limit interactions with the management layers of the RTI, by further reducing the computation and communication overheads.

To sum up, by limiting the network communication in favour of the local (shared memory) communication, the wall clock time required by the simulation run-time to achieve full synchronization would be reduced. This would make it possible to obtain a fast simulation.

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 modeller, on the basis of the modeling assumptions. In addition, as it is demonstrated in figure 2, the model dynamics (e.g. the SMH mobility) would make useless the optimal initial distribution after few simulation steps. This result may translate in a performance degradation for the simulation speedup, mainly due to the increasing cost of communication and synchronization required between distributed model components (logical processes).

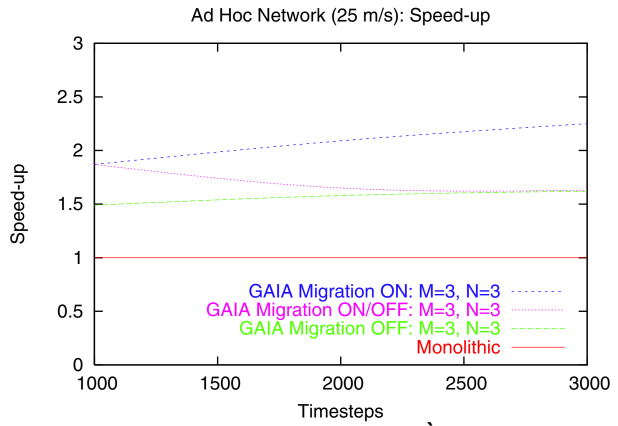

Figure 2. Effect of GAIA over ARTÌS simulation

The curves "GAIA Migration ON" and "GAIA Migration OFF" show the runtime speedup obtained by distributed implementations with respect to a monolithic simulation of the wireless system model. It is worth noting that the optimal runtime allocation and balancing obtained with GAIA gives better results (N=3 LPs are simulated over M=3 PEUs). The curve "GAIA Migration ON/OFF" shows the effect of the initial static optimization: under the effect of the model dynamics (without the GAIA runtime optimization) the performance converges to the suboptimal performance in few timesteps. In GAIA the optimization is dynamically performed at run-time, by the proposed simulation middleware migrating the SMHs between LPs. In this way, the modeller is alleviated by the optimization task, and the system converges towards a balanced, tuneable and pseudo-optimal model components' distribution driven by the model interaction scheme.

5. Concurrent replication of parallel and distributed simulation (CR-PADS)

A new direction for trying to maximize the speedup of the simulation processes and the utilization of computation (and communication) resources in the system architecture supporting the parallel or distributed simulation is based on the Concurrent Replication of Parallel and Distributed Simulations (CR-PADS) [6]. 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. To satisfy this requirements we choose to modify our ARTÌS simulation framework, by adding a module which is able to implement CR-PADS, rather than executing a sequence of multiple parallel or distributed simulation runs.

The replication concept 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 an 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 only at the beginning of the simulation process (differently from cloning schemes, creating replicas at runtime). Each replica is an independent run, with its own seeds, and model initialization. The CR-PADS approach is different from the Multiple Replication In Parallel approach (MRIP) where a number of sequential runs are executed in parallel over the available set of PEUs (see figure 3.a). In the following we are going to explain the motivations for our proposal of a Concurrent Replication mechanism for Parallel and Distributed Simulation (CR-PADS) by sketching the differences among the MRIP, the PADS and the CR-PADS approaches, under both the computation and communication viewpoints. In figure 3, by assuming that N CPUs are made available for computation (no matter if they belong to parallel or distributed architectures) we want a set of many independent runs to be executed (only two in the figure for clarity). 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 (figure 3.a) 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 speedup, 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, as the figure 3.a illustrates, if the available resources are more than the number of runs required, the potential associated to some resources may remain unexploited.

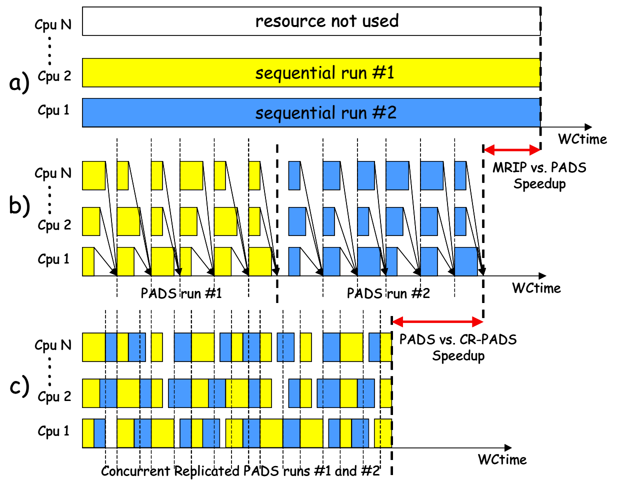

Figure 3. A comparison of MRIP, PADS and CR-PADS

The parallel or distributed simulation (PADS) approach (see figure 3.b) 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 in the scenario of figure 3.b may result in a speedup depending on the number of resources and the number of runs required under MRIP and PADS, respectively. With PADS, the advantages of the aggregate memory architecture may assist the model data structures management, and the whole set of computation resources (CPUs) can be exploited in parallel. The problem arising under the PADS scenario is represented in the figure 3.b: 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 advances 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 in figure 3.b swings between computation and idle periods, while the underlying communication infrastructures swings between idle and communication periods, respectively.

In [6] we showed it is possible 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 (figure 3.c). 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 are smoothed, and this may result in additional speedup with respect to the whole time required for completing the whole 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 live-locks, resulting in trashing effects. 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.

We defined the structure of the ARTÌS framework 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, based on queue management, priority and fairness protocols.

The management of Random Numbers Generators (RNGs) in CR-PADS is simpler than in cloning approach, because seeds can be chosen at the beginning of the runs. Figures 4 and 5 show the time required to complete the x concurrent simulation runs (indicated by X values) for the wireless ad hoc network model composed by 1000 SMHs, and N LPs executed over M PEUs. Shared memory, and LAN-based PEU architectures, are considered in figures 4 and 5, respectively. In figure 4, the CR-PADS parallel executions over tightly coupled (shared memory) PEUs gives advantages (with respect to PADS) with more than one concurrent replication, and introduces overheads when the number of replicas saturates the computation power and/or communication bandwidth of the PEU execution architecture. In distributed scenarios (LAN-based communication, see figure 5), the CR-PADS gives even better results, because the communication latency is the bottleneck parameter in the execution architecture, and more concurrent replicas can exploit more residual computation power [6].

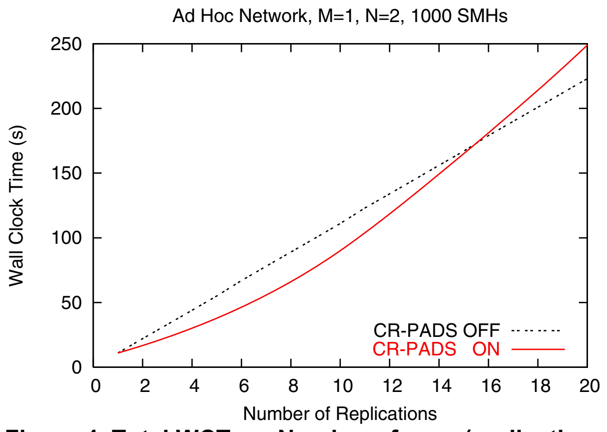

Figure 4. Total WCT vs. Number of runs (replications) Parallel simulation scenario: M=1, N=2, 1000 SMHs

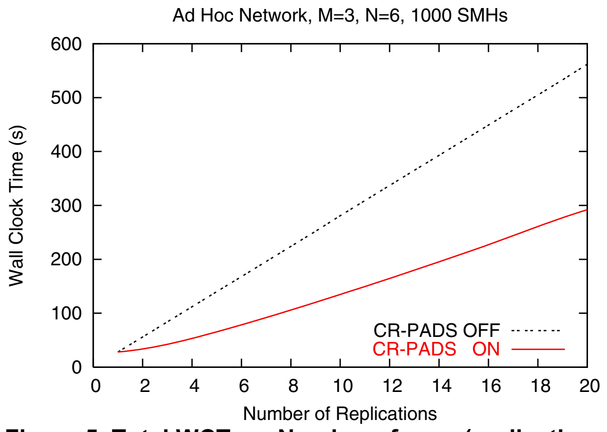

Figure 5. Total WCT vs. Number of runs (replications) Distributed simulation scenario: M=3, N=6, 1000 SMHs

6. Communication and computation optimization for PADS

To the best of our knowledge, the influence of Hyper-Threading (HT) technology on PADS architectures and frameworks has not been investigated in detail. Intel's HT technology is a new architectural solution for support of virtual concurrency in the processes' execution [7]. Thanks to simple heuristics, HT-enabled OSes should be able to adapt the process scheduling to the HT architecture, with the aim of optimizing the overall execution of processes.

On the application side, it is quite common for parallel and distributed simulation frameworks to allocate a single

LP for each available processor. This is based on the assumption that one single LP will be the only running process and would not cause context switches and other relevant overheads. On the other hand, each time the LP is blocked due to communication and synchronization delays, the CPU time would be wasted [6]. The effects of HT technology could significantly change the assumptions related to current implementation choices.

Under the software architecture viewpoint, most of the modern PADS middlewares are based on multi-threaded implementation, and basically should take advantage of the HT support. It would be interesting to evaluate if the increasing in the number of logical processes could be exploited as a new dimension for PADS optimization: to concurrently run more LPs than the number of physical processors, over HT processor architectures.

To give answers to the above questions about HT technology and PADS assumptions, we evaluated the performances of the real ARTÌS simulation framework on a real experimental testbed, instead of relying on synthetic CPU benchmarks.

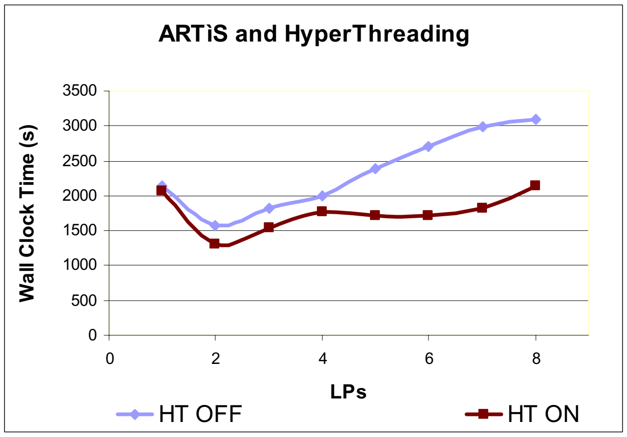

Figure 6: WCT obtained with Hyper-Threading support On and OFF in a PADS simulation of 6000 model entities, as a function of the number of LPs

The wireless mobile ad hoc network model of a complex and dynamic system with 6000 SMHs, was assumed for the analysis. The experiments have been executed over a PEU equipped by Dual Xeon Pentium IV 2800 MHz, 3 GB RAM. The first group of tests is based on Hyper-Threading support enabled for the PEU (HT-ON), and the second one is based on Hyper-Threading support disabled directly by BIOS settings (HT-OFF).

In Figure 6, results show that the number of HT-enabled PEU (HT-ON) indicated on the X axis performs always better than the HT-disabled (HT-OFF) version. In addition, the HT-ON scenario shows an increasing simulation scalability, with respect to HT-OFF.

In ARTÌS, a reduction of the overheads and channel accesses could result in increased channel utilization and reduction of the communication bottlenecks. For this reason we investigated if and how a message marshalling approach could reduce the simulation WCT.

The data marshalling approach consists in the concatenation of more than one logical message in the same communication messages. In order to control the degradation in the average communication latency, the data marshalling process is controlled by a timer: once every a maximum time limit the messages buffered on the LP are sent in a data marshalling packet (or frame). The proposed optimization has been applied both to shared memory and TCP/IP communications. Figure 7 shows the results for the ARTÌS optimization applied to a distributed simulation architecture when executing a PADS simulation of 3000, 6000 and 9000 model entities in the wireless ad hoc network model.

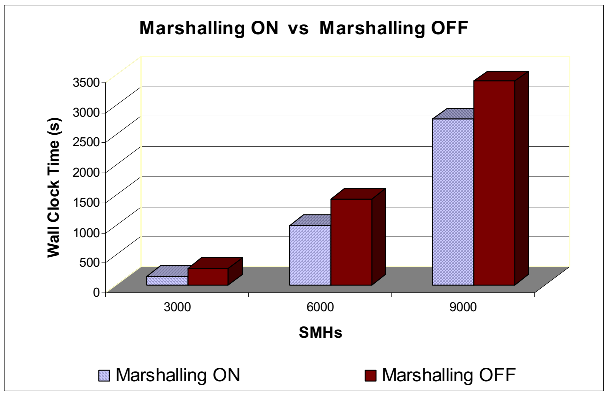

Figure 7: WCT obtained with Marshalling On/OFF in a PADS simulation of 3000, 6000 and 9000 model entities

7. Conclusions

In this work we illustrated the design and implementation guidelines of the ARTÌS middleware defined to support the parallel and distributed simulation of large scale, complex and dynamically interacting system models. We designed and implemented the ARTÌS middleware by incorporating a set of features that allow adaptive optimization by exploiting many complex and dynamic model and distributed simulation characteristics. As an example, a dynamic migration mechanism (GAIA) for the run-time adaptive allocation of model entities has been designed and exploited for dynamic load and communication balancing. Optimizations have been introduced to obtain the maximum advantage from heterogeneous and asymmetric communication systems, from shared memory to LAN and Internet-based communication. Other optimizations have been introduced by the exploitation of concurrent replications of parallel and distributed simulations, in order to increase the resources utilization and to maximize the speedup of simulation processes. Solutions have been designed, implemented and tuned to obtain a significant reduction in the communication and synchronization overheads between the physical execution units, and an increased model scalability and simulation speedup, even in worst-case modeling assumptions and simulation scenarios. Other solutions have been proposed to improve the performance of simulations executed over commodity off-the-shelf computation and communication architectures: multi-threaded software and Hyper-Threading support by the processor architectures, data marshalling solutions for shared-memory and network-based communications. All the proposed solutions have been evaluated on real testbed evaluation scenarios, and under variable configurations. Results obtained demonstrate that a performance improvement can be obtained by adopting and tuning the proposed solutions in opportune way. The experimental analysis performed in [3, 4, 5, 6, 7] has provided some interesting guidelines about the way to adopt and to compose the proposed solutions in ARTÌS.

References

[1] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, and H.Y. Song, "PARSEC: a parallel simulation environment for complex systems", IEEE Computer, 31(10), October 1998, pp.77-85

[2] A. Berrached, M. Beheshti, O. Sirisaengtaksin, and A. Korvin, "Alternative Approaches to multicast group allocation in HLA data distribution", Proc. Of the 1998 Spring Simulation Interoperability Workshop, 1998

[3] L.Bononi, G.D'Angelo, L.Donatiello, "HLA-Based Adaptive Distributed Simulation of Wireless Mobile Systems", in Proceedings of IEEE/ACM Intern. Workshop on Parallel and Distributed Simulation (PADS'03), San Diego, CA, June 2003

[4] L.Bononi, M.Bracuto, G.D'Angelo, L. Donatiello, "ARTÌS: a Parallel and Distributed Simulation Middleware for Performance Evaluation", in LNCS 3280 - 2004 Proceedings of the 19-th International Symposium on Computer and Information Sciences (ISCIS 2004), Kemer-Antalya, Turkey, October 27-29, 2004, pp. 627-637

[5] L.Bononi, M.Bracuto, G.D'Angelo, L. Donatiello, "A New Adaptive Middleware for Parallel and Distributed Simulation of Dynamically Interacting Systems", in Proceedings of the 8-th IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2004), Budapest, Hungary, October 21-23, 2004

[6] L.Bononi, M.Bracuto, G.D'Angelo, L.Donatiello "Concurrent Replication of Parallel and Distributed Simulation", Proceedings of the 19th ACM/IEEE/SCS International Workshop on Principles of Advanced and Distributed Simulation (PADS 2005), Monterey, CA, USA, June 2005

[7] L.Bononi, M.Bracuto, G.D'Angelo, L. Donatiello, "Analysis of high performance communication and computation solutions for parallel and distributed simulation", submitted for publication, 2005

[8] A. Boukerche, and S.K. Das, "Dynamic Load Balancing Strategies for Conservative Parallel Simulation", Proc. of 11-th Workshop on Parallel and Distributed Simulation (PADS'97), June 1997, Lockenhaus, Austria, pp. 20-28

[9] A. Boukerche, and A. Fabbri, "Partitioning Parallel Simulation of Wireless Networks", Proc. of the 2000 Winter Simulation Conference (WSC), 2000

[10] K.M.Chandy and J.Misra, "Distributed simulation: a case study in design and verification of distributed programs", IEEE transactions on Software Engineering, 5(5): 440-452, 1979

[11] J. Dahmann, R.M. Fujimoto, and R.M. Weatherly, "High Level Architecture for Simulation: an update", Winter Simulation Conference, December 1998

[12] S.R. Das, "Adaptive protocols for Parallel Discrete Event Simulation", Proc. of Winter Simulation Conference, 1996

[13] W.J. Davis, G.L. Moeller, "The High Level Architecture: is there a better way?", proc. Winter Simulation Conference, 1999

[14] E. Deelman, and B.K. Szymanski, "Dynamic load balancing in parallel discrete event simulation for spatially explicit problems", Proc. of the 12-th workshop on Parallel and distributed simulation PADS'98, July 1998

[15] DMSO: Defence Modeling and Simulation Office (1998), High Level Architecture RTI Interface Specification, Vers. 1.3

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

[17] Fujimoto, R.M., Parallel and Distributed Simulation Systems, John Wiley & Sons, 2000

[18] Georgia Tech RTI-kit, http://www.cc.gatech.edu/computing/pads/index.html, 2005

[19] P. Huang, D. Estrin, and J. Heidemann, "Enabling large-scale simulations: Selective abstraction approach to the study of multicast protocols", proc. Mascots'98, Oct. 1998

[20] 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), 2000

[21] D.R. Jefferson, "Virtual Time", ACM Trans. Prog. Lang. and Syst. 7(3): 404-425, 1985

[22] B. Logan, and G. Theodoropoulos, "The Distributed Simulation of Multi-Agent Systems", Proc. of the *IEEE*, 2001

[23] D.M. Rao, and P.A. Wilsey, "An Ultra-large Scale Simulation Framework", Proc. of MASCOTS '99, Oct. 1999

[24] D.M. Rao, and P.A. Wilsey, "An object oriented framework for parallel simulation of ultra-large communication networks", proc. 3-rd Inter.l symposium on computing and object oriented parallel environments, Nov. 1999

[25] D.M. Rao, and P.A. Wilsey, "Parallel Co-simulation of Conventional and Active Networks", Proc. of MASCOTS'00, August 2000

[26] G.F. Riley, R.M. Fujimoto, M.H. Ammar, "A generic framework for parallelization of network simulations", Proc. of MASCOTS'99, College Park, MD, October 1999

[27] G.F. Riley, and M.H. Ammar, "Simulating Large Networks How Big is Big Enough?", Proc. of First Intern.l Conference on Grand Challenges for Modeling and Simulation, Jan. 2002

[28] J. Short, R. Bagrodia, and L. Kleinrock, "Mobile wireless network system simulation", Wireless Networks 1, August 1995

[29] T.K. Som, and R.G. Sargent, "Model structure and load balancing in optimistic parallel discrete event simulation", Proc. of the 14-th workshop on Parallel and distributed simulation, May 2000, Bologna

[30] B.K. Szymanski, and Y. Liu, "Loosely-coordinated, distributed, packet-level simulation of large-scale networks", Proc. of the Winter Simulation Conference, December 2003

[31] V-Y Vee, and W-J Hsu, "Locality-preserving load-balancing mechanisms for synchronous simulations on shared-memory multiprocessors", Proc. of 14-th PADS 2000, Bologna, Italy, page 131-138