Concurrent Replication of Parallel and Distributed Simulations

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

Parallel and distributed simulations enable the analysis of complex systems by concurrently exploiting the aggregate computation power and memory of clusters of execution units. In this paper we investigate a new direction for increasing both the speedup of a simulation process and the utilization of computation and communication resources. Many simulation-based investigations require to collect independent observations for a correct and significant statistical analysis of results. The execution of many independent parallel or distributed simulation runs may suffer the speedup reduction due to rollbacks under the optimistic approach, and due to idle CPU times originated by synchronization and communication bottlenecks under the conservative approach. We present 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. Results obtained from tests executed under variable scenarios show that speedup and resource utilization gains could be obtained by adopting the proposed replication approach in addition to the pure parallel and distributed simulation.

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 complex model definition and dynamic interactions among a possibly huge set of model components. The accuracy of simulation results requires a detailed definition and simulation of every single model component of the whole complex system model. To overcome the computation and memory bottlenecks of mono-processor architectures [22, 23], many practical experiences have demonstrated that a solution for the simulation of such systems is achievable by using parallel and distributed models and architectures, i.e. a Parallel or Distributed Discrete Event Simulation (PDES) approach [14, 16]. The IEEE 1516 Standard for Modeling and Simulation High Level Architecture (HLA) is a recently approved standard dealing with component-oriented distributed simulation [7, 12]. It defines rules and interfaces allowing for heterogeneous model components' interoperability in distributed simulation.

In presence of a complex dynamic system, the implementation of a parallel or distributed simulation would lead to synchronization and communication overheads [3, 4]. The research on the optimization and overheads reduction in parallel and distributed simulations has gained a great interest, leading to the design and implementation of parallel and distributed models and simulation frameworks [3, 10, 17, 22, 24]. In general, an inverse tradeoff exists that determines a mutual worsening in the speedup: i) by reducing the degree of parallelism in the computation or, conversely, ii) by the communication bottlenecks and blocking synchronization primitives among many heterogeneous simulation components. 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 speedup, both in optimistic and conservative approaches [8, 10, 17]. Adaptive behavior in the management of the model at runtime, to control the overheads due to the model dynamics, have been considered in [3]. A perfect load balancing over 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 rapidly executed components (Logical Processes, LPs) 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 (by wasting local CPU time), and the whole simulation process proceeds with the speed of the slowest components. The frequent synchronizations are usually implemented as message passing primitives, and may be heavily affected by the overheads for communication management and communication bottlenecks.

In this paper we investigate a new direction for trying to maximize the speedup of the simulation processes and the

This work is supported by MIUR FIRB funds, under the project: "Performance Evaluation of Complex Systems: Techniques, Methodologies and Tools"

Proceedings of the Workshop on Principles of Advanced and Distributed Simulation (PADS'05)
1087-4097/05 $20.00 © 2005 **IEEE**

utilization of computation and communication resources in the system architecture supporting the parallel or distributed simulation. Our assumption, based on a common analysis rule, is that a simulation-based analysis requires many independent parallel or distributed simulation runs, to collect 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 [11, 13, 15, 20, 21, 22, 24] 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 common model definition, and realizes an independent execution (i.e. replicas do not require mutual synchronization) based on local initial parameters, variable factors for the analysis, and different random number generation seeds. The CR-PADS is intended as a way to maximize the speedup and utilization of system resources when implementing a set of parallel and distributed simulations of complex dynamic system models. This approach is substantially different from the simulation Cloning concept [6, 18]. Simulation cloning has been demonstrated as a good technique for supporting "faster than real time" simulation [18] and "what if" analysis [6] being based on the active cloning of multiple instances of the initial simulation process at "decision points" that may be met during the simulation. 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. The CR-PADS replication approach is also different from the Multiple Replications in Parallel (MRIP) approach adopted for the concurrent execution of independent sequential simulation runs, like in the Akaroa2 framework [1, 2, 11, 13, 15, 22]. The aim of MRIP in Akaroa2 is basically to give a simple way to the modeller for initiating multiple independent runs of sequential simulators over different processors. To the best of our knowledge, 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. With respect to CR-PADS approach, Akaroa2 and the MRIP approach may have limitations in exploiting the aggregate resources of a cluster of execution units, and the adoption of distributed models, which are some of the motivations in favour 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 [5]. The ARTÌS framework is a middleware currently under implementation, whose design is inspired to the High Level Architecture (HLA) design and IEEE 1516 Standard. Results obtained for the simulation of a complex dynamic system model (i.e. a wireless ad hoc network model) demonstrate that a speedup gain can be obtained by adopting the proposed replication approach as an alternative to a sequence of standalone parallel and distributed simulations. The speedup 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 paper structure is the following: in section 2 we outline some concepts about the parallel and distributed simulation cloning and replication; in Section 3 the key issues for the replication mechanism implementation and the ARTÌS middleware are defined; in section 4 a prototype wireless system's model and a set of simulation results are presented to evaluate the concurrent replication approach; in section 5 we summarize our conclusions and future work.

2. Cloning and Replication of Parallel and Distributed Simulations

The Multiple Replications in Parallel (MRIP) is a technique that we cite here as a reference for our work [1, 2, 11, 13, 15, 20, 21]. 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++ [13, 22]. To the best of our knowledge, the MRIP approach for parallel and distributed simulations is still to be investigated.

2.1. 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. 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 [19], 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, 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 conservative 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.

2.2. 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, 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 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 interconnection 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 clear that local IPC is preferable (if available) to be exploited over parallel architectures.

2.3. Parallel and Distributed Simulation Cloning

The simulation cloning technology was introduced as a concurrent evaluation mechanism, in the context of parallel simulation [18]. 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.

The cloning approach includes the management of different time axes in parallel, in order to support a runtime forecasting functionality [24]. In [6] the aim to support users willing to run existing complex simulation models, gave to 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 [6, 18]. A preliminary discussion of the design and implementation challenges and solutions,

like virtual logical processes (VLPs) and virtual messages (VMs), can be found in [6, 18]. The design and implementation of runtime support for cloning of HLA-based parallel and distributed simulations is quite complex and challenging effort. This is even more critical under HLA-based and Data Distribution Management (DDM) based implementations.

2.4. The Concurrent Replication of Parallel and Distributed Simulations

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. Each replica is based on a common model definition, and realizes an independent run 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.

2.4.1. Motivations for the Concurrent Replication. 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 PDES and the CR-PADS approaches, under the computation and communication viewpoints.

In figure 1, by assuming that a set of 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 1a), 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 1a illustrates, if the available resources are more than the number of runs required, the potential associated to some resources may remain unexploited.

The parallel (or distributed) discrete event simulation (PDES) approach (see figure 1b) 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 1b may result in a speedup 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.

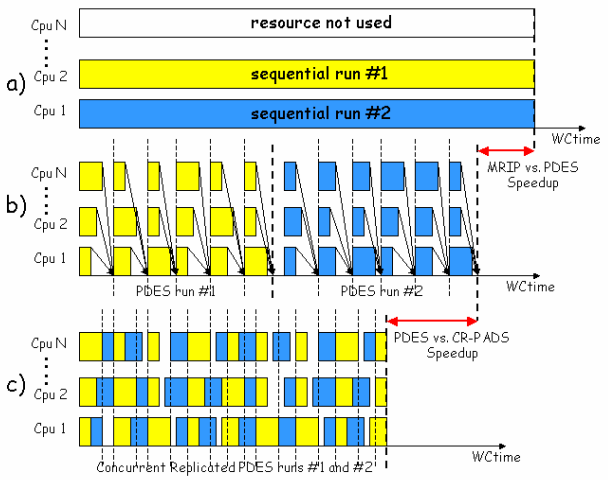


Figure 1. Comparing MRIP vs. PADS vs. CR-PADS

The problem arising under the PDES scenario is represented in the figure 1b, where 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 may result in a high percentage of the whole synchronization delay, under loosely coupled distributed architectures (e.g. over CPUs interconnected by LAN or Internet technology). In other words, as shown in figure 1b, every simulation process may result in a sequence of mixed and interleaving bottleneck phases originated i) by the CPU computation and ii) by synchronization and communications, respectively.

In this work we investigate how 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. By launching multiple, independent and concurrent parallel or distributed simulation runs, idle CPU time could be avoided by switching to the execution of computation requests by the other replicas which already completed their synchronization phase. The same principle could be exploited 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 phases, and idle or congested communication phases, could be smoothed over time. This may result in additional speedup with respect to the 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 livelocks, 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.

2.4.2. Advantages of the Replication approach. Among the advantages of a concurrent replication approach for parallel and distributed simulations, at the top layer we identify the possible speedup obtainable when executing a set of many independent runs, or the possible concurrent analysis of different scenarios (defined at the beginning of the run). Specific implementation guidelines could lead to additional advantages. As an example, we defined the structure of the ARTÌS framework such that a separation of the simulation and replication management is specifically devoted to a clean design and to the exploitation of management techniques that reduce the communication overheads.

Replicas are created 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) is simpler than in the cloning approach, 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.

3. The CR-PADS Implementation

Recently proposed and implemented middleware solutions based on the IEEE 1516 Standard have shown that the parallel and distributed simulation of massive and complex systems can result in relevant overheads [3, 4]. Overheads are due to the complex and full management of a wide set of runtime services and to the latency due to distributed communication bottlenecks. Specifically, the implementation of the interprocess communication services has been implemented in sub-optimal way, without considering the heterogeneity of the simulation execution platforms [3, 4].

The HLA implementation criticisms [3, 4, 7, 9] and the lack of efficient Open Source runtimes are the main motivations behind the design and implementation of a new parallel and distributed simulation middleware named Advanced RTI System (ARTÌS) [5]. The aim of the ARTÌS middleware is to support parallel and distributed simulations of complex systems, based on a minimal set of efficiency-oriented middleware services. The 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. Another design issue of the ARTÌS framework is the dynamic adaptation of the interprocess communication layer to the heterogeneous communication support offered by possibly different simulation-execution units. Specifically, the ARTÌS design is based on the adaptive evaluation of the communication bottlenecks and support for multiple communication infrastructures and services, from shared memory to Internet-based communication.

The ARTÌS implementation follows a component-based design, that results in easily extendable middleware (see figure 2a). The solutions proposed for time management and synchronization in distributed simulations have been widely analyzed and discussed in the design phase. Currently, ARTÌS supports the conservative time management based on both the time-stepped approach, and the Chandy-Misra-Bryant (CMB) algorithm. The extension of ARTÌS to support optimistic time management algorithms (Time Warp) is ongoing work.

3.1. Implementation of Replication in ARTÌS

Given the top level structure of ARTÌS shown in figure 2a, ARTÌS supports the execution of LPs over the RTI kernel. The RTI kernel implements time management policies, object ownership and 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. Specifically, communication is implemented over shared memory for parallel processors, over Reliable-UDP for LANs and TCP/IP for Internet-based communication.

In ARTÌS, many design optimizations have been applied to obtain adequate protocols for synchronization and

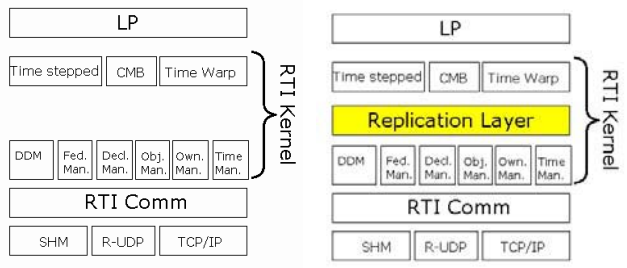
communication over Local Area Networks (LAN) or Shared Memory (SHM) multiprocessor architectures. In our vision the communication and synchronization middleware should be adaptive and user-transparent about all the optimizations required to improve performances. The current scheme adopts an incremental straightforward policy: given a set of LPs on the same physical host, such processes always communicate and synchronize via read and write operations, performed within the address space of LPs, in the shared memory (see PEU X on figure 3). To implement these services we have designed, implemented and tested many different solutions, based on inter-process communication (IPC) semaphores and locks, busy-waiting, and "wait on signals" with a limited set of temporized spin-locks. The latter solution has demonstrated very low latency and limited CPU overhead, good performances obtained in multi-CPU systems, good scalability, and no need to reconfigure the operating system kernel level.

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. Two or more LPs located on Internet hosts rely on standard TCP/IP connections (see PEUs X and Y in figure 3).

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 is waiting for messages on each channel, and demultiplexes 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 layers would have given benefits under the optimization viewpoint, at the additional expenses of ad hoc re-implementation of the replication services for each time management policy.

The Replication management layer is the funnel for replicas over the RTIComm layer (see figure 2b). The Replication layer generates the replicas of each LP, and manages 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. UNIX pipes have been preferred to shared data structures, because messages have limited size and the consistency management of shared data structures would introduce latency and additional overheads.



a b

Figure 2. The ARTÌS and Replication architecture

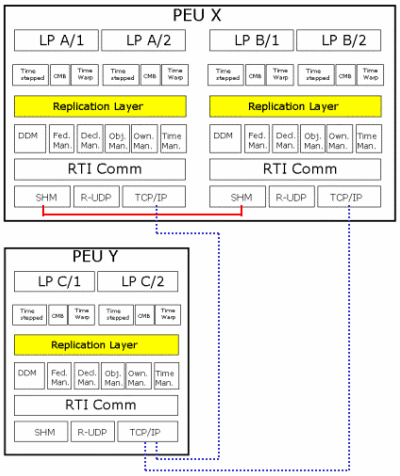


Figure 3. Parallel and distributed CR-PADS structure

The proposed implementation architecture allows optimization of the communication management locally to the LPs. Alternative architectures aiming to reduce the communication bottleneck are currently under evaluation. In the following, we sketch the dynamic behavior of the Replication management layer when replicating 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 freed.

4. Performance Evaluation

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

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. Specifically, the execution architecture is composed by three homogeneous units defined like the aforementioned PEU.

As a testbed for our replication framework, we 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 has the effect of controlling 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 paper, we 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.

4.2. Performance results

In this section we present 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.

We 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 we define 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 4, 5 and 6 we report results for the parallel simulation environment (M=1, N=2). Each figure shows the Wall Clock Time (WCT) required for completing the *x* simulation processes. Figure 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 and 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 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 (fig. 5) and 2000 (fig. 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.

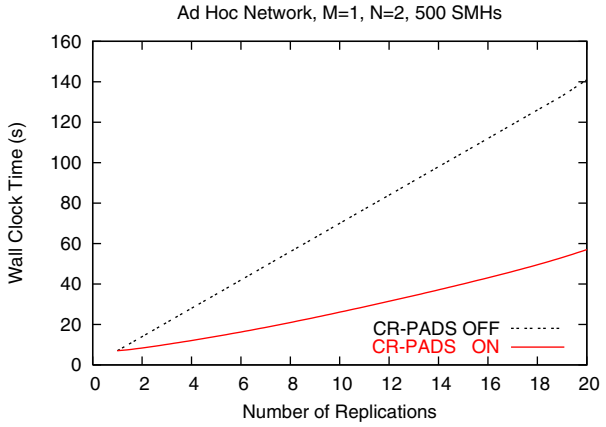


Figure 4. Total WCT vs. Number of runs (replications) Parallel simulation scenario: M=1, N=2, 500 SMHs

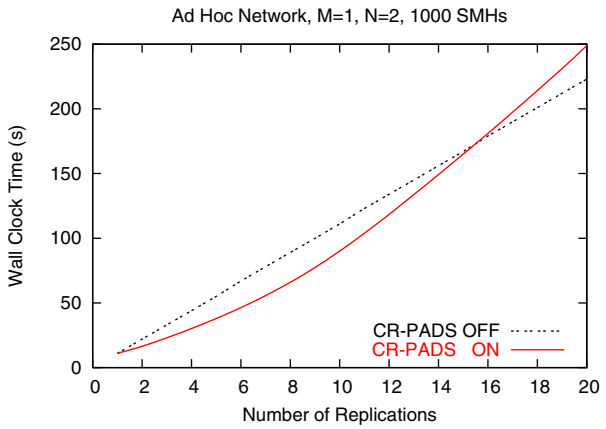


Figure 5. Total WCT vs. Number of runs (replications) Parallel simulation scenario: M=1, N=2, 1000 SMHs

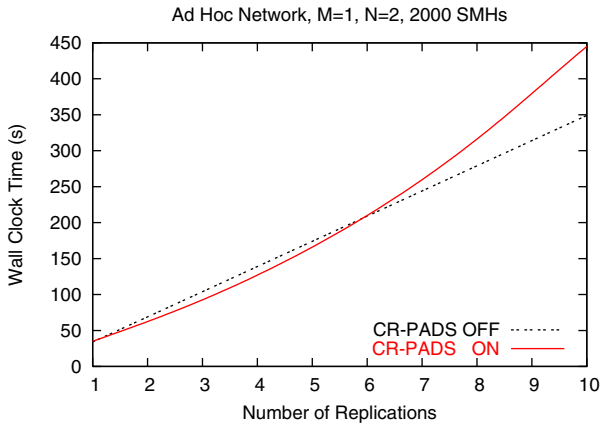


Figure 6. Total WCT vs. Number of runs (replications) Parallel simulation scenario: M=1, N=2, 2000 SMHs

It is worth noting that the break-even in the number of replicas is around 15 in figure 5 and around 6 concurrent replicas in figure 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 speedup 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 7, 8 and 9 we illustrate 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.

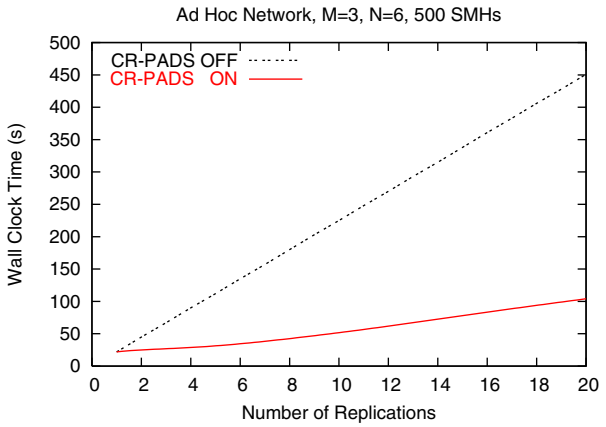


Figure 7. Total WCT vs. Number of runs (replications) Distributed simulation scenario: M=3, N=6, 500 SMHs

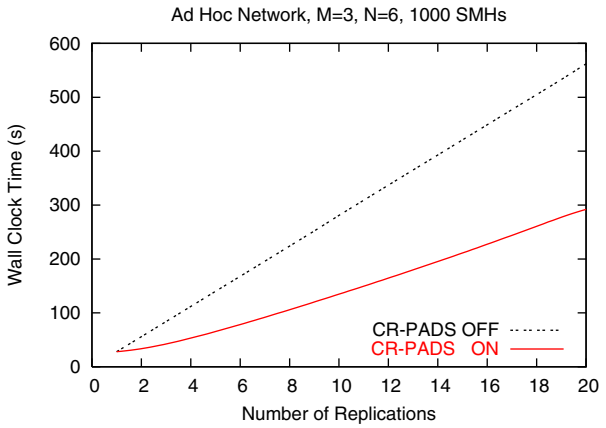


Figure 8. Total WCT vs. Number of runs (replications) Distributed simulation scenario: M=3, N=6, 1000 SMHs

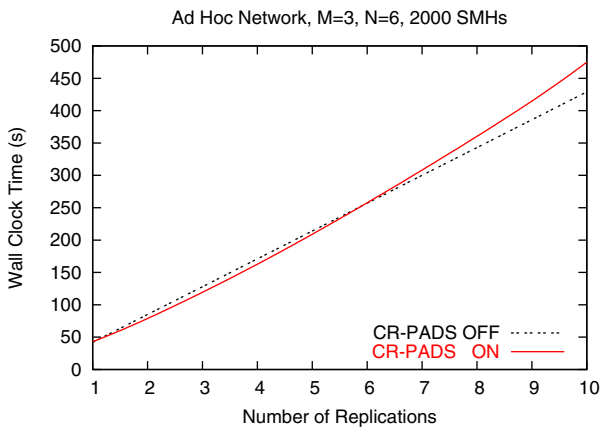


Figure 9. Total WCT vs. Number of runs (replications) Distributed simulation scenario: M=3, N=6, 2000 SMHs

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 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 speedup both in the 500 and 1000 SMHs scenarios (figures 7 and 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 does not introduce significant overheads in the range of 1 up to 10 concurrent replications.

Figure 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 10, CR-PADS allows a high event computation density when the computation load in each timestep does 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 computation concurrency in legacy distributed simulations. As expected, the CR-PADS mechanism is able to maintain a high computation concurrency.

Figure 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 converge 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 performed, which is the system bottleneck in the current scenario).

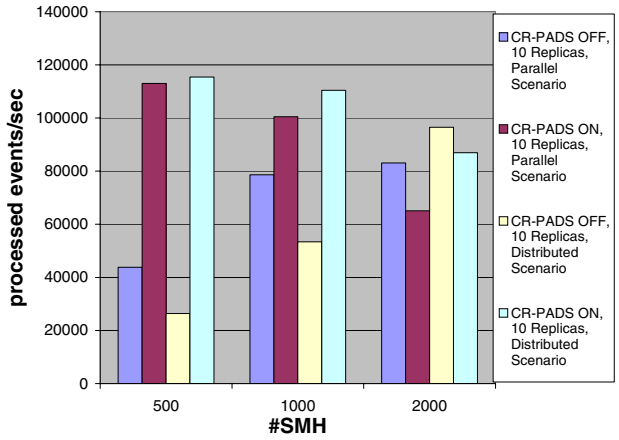


Figure 10. Analysis of event processing rate

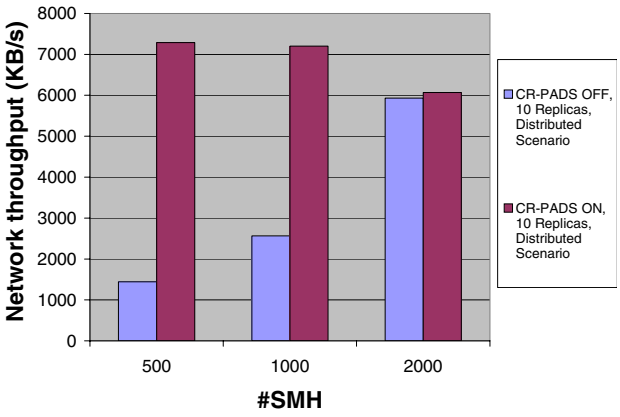


Figure 11. Analysis of communication throughput

5. Conclusions and future work

In this paper we propose and investigate a new direction for increasing both the speedup of a parallel or distributed 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 speedup reduction due to rollbacks under the optimistic approach, and due to idle CPU times originated by synchronization and communication bottlenecks under the conservative

approach. We present 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. The implementation of the CR-PADS mechanism has been defined and tested over the Advanced RTI System (ARTÌS) framework, which is inspired to the HLA architecture. Results obtained from tests executed under variable scenarios show that speedup gains could be obtained by adopting the proposed replication approach in addition to the pure parallel and distributed simulation.
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, and with the components for runtime transient and steady-state analysis of data, confidence interval estimation, and run termination management.

6. References

[1] D. Anagnostopoulos and M. Nikolaidou, "Executing a Minimum Number of Replications to Support the Reliability of FRTS Predictions", proc. 7-th IEEE DS-RT 2003, Delft, The Netherlands, Oct. 2003

[2] W. Biles and J.P.C. Kleijnen, "Statistical Methodology for WEB-Based Simulation", proc. 7-th IEEE DS-RT 2003, Delft, The Netherlands, Oct. 2003

[3] L. Bononi, G. D'Angelo and L. Donatiello, "HLA-Based Adaptive Distributed Simulation of Wireless Mobile Systems", in Proceedings of IEEE/ACM Int'l Workshop on Parallel and Distributed Simulation (PADS'03), San Diego, CA, 06/2003

[4] L. Bononi, M. Bracuto, G. D'Angelo and L. Donatiello, "A New Adaptive Middleware for Parallel and Distributed Simulation of Dynamically Interacting Systems", proc. 8-th IEEE Int'l Symposium on Distributed Simulation and Real Time Applications (DS-RT 2004), Budapest, Hungary, Oct. 2004

[5] L. Bononi, M. Bracuto, G. D'Angelo and L. Donatiello, "ARTIS: a Parallel and Distributed Simulation Middleware for Performance Evaluation", in LNCS 3280 - 2004 Proc. of the 19-th International Symposium on Computer and Information Sciences (ISCIS 2004), Antalya, Turkey, Oct. 2004, 627-637

[6] D. Chen, S.J. Turner, B.P. Gan, W.T. Cai, J. Wei and N. Julka "Alternative Solutions for Distributed Simulation Cloning", Simulation, Vol. 79, No. 5-6, 299-315 (2003)

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

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

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

[10] 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

[11] E. de Souza Mota, K. Pawlikowski and A. Wolisz. "A Perspective of Batching Methods in Simulation Environment of Multiple Replications in Parallel", Proc. Winter Simulation Conf. 2000, Orlando, Florida, USA, Dec. 2000, pp. 761-766

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

[13] G. Ewing, K. Pawlikowski and D. McNickle. "Akaroa2: Exploiting Network Computing by Distributing Stochastic Simulation", Proc. European Simulation Multiconference ESM'99, Warsaw, ISCS, June 1999, pp. 175-181

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

[15] F.Fitzek, E. Mota, E. Ewers, A. Wolisz, and K. Pawlikowski, "An Efficient Approach For Speeding Up Simulation Of Wireless Networks", In Proc. of WMC 2000, San Diego, California, USA, January 2000.

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

[17] B.P. Gan, Y.H. Low, S. Jain, S.J. Turner, W. Cai, W.J. Hsu and S.Y. Huang, "Load balancing for conservative simulation on shared memory multiprocessor systems", Proc. of the 14-th workshop on Parallel and distributed simulation (PADS'00), May 28-31, 2000, Bologna, Italy, p.139-146

[18] M. Hybinette and R.M. Fujimoto. "Cloning parallel simulations". ACM Trans. Model. Comput. Simul., 11(4), 2001.

[19] D.R. Jefferson, "Virtual Time", ACM Trans. on Prog. Languages and Systems, Vol.7 No.3, pp. 404-425, 1985

[20] K.G. Jones and S.R. Das, "Parallel Execution of a sequential network simulator", Proc. of the 2000 Winter Simulation Conference, 2000

[21] Y.B. Lin, "Parallel independent replicated simulation on a network of workstations", In proc. of the 8-th workshop on Parallel and distributed simulation, pp. 73--80, 1994.

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

[23] 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

[24] T. Schulze, S. Straßburger, and U. Klein, "HLA- federate reproduction procedures in public transportation federations" proceedings of the 2000 Summer Computer Simulation Conference, July, Vancouver, Canada