NetStep: a micro-stepped distributed network simulation framework

Olivier Dalle
Laboratoire I3S UMR CNRS 7271
INRIA Sophia Antipolis - Mediterranée
France
olivier.dalle@inria.fr

Emilio P. Mancini
Laboratoire I3S UMR CNRS 7271
INRIA Sophia Antipolis - Mediterranée
France
emilio.mancini@inria.fr

ABSTRACT

This paper presents NetStep, a prototype for the distributed simulation of very large scale network simulations, such as the simulation of peer-to-peer applications. We use simulation micro-steps as a means for optimizing the overlap of communications and computations, without changing the original event-driven model. As a consequence, NetStep allows for the reuse of unmodified existing sequential simulators for building large-scale distributed simulations: the overall simulation is divided both in time and space, into a large number of simulation micro-steps, each of which being executed by a legacy sequential simulator. By choosing the time-step smaller than the minimal look-ahead due to communications, we avoid the need for synchronization between Logical Processes during the simulation. Instead, the simulated communications become inputs and outputs of the simulation micro-steps, and are routed in parallel between LPs by a NetStep dedicated entity. Our prototype is based on the SimGrid sequential simulator.

Categories and Subject Descriptors

I.6.8 [**Simulation and Modeling**]: Types of Simulations—*parallel*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; D.2.13 [**Software Engineering**]: Reusable Software

General Terms

Simgrid, Peer-to-peer computing

Keywords

time-step, overlap, look-ahead, simulator

1. INTRODUCTION

Analytical methods, especially the stochastic ones, are commonly used to study large distributed systems like peer-to-peer systems [8, 5]. Unfortunately, such techniques require to comply with a number of hypotheses (independence of events, steady-state, stationarity, etc.) that make them unsuitable for the study of many problems. Given that experimenting on real systems have an obvious hard limit in terms of scalability, the only option left for the study of such very large systems is simulation. However, unless the model of the system is over-simplified, a parallel approach seems unavoidable to cope with the resulting computation complexity of the simulation. This complexity can easily be explained by the two-dimensional nature of the simulation models: The longer the history of the model needs to be computed, the longer the computation needs to run, and the larger the model space (usually abstracted in terms of states), the more the computation needs memory.

If we except the special case for parallel replications, parallelization techniques follow this two-dimensional scheme in time and space[4] and fall accordingly into the categories of time-parallel and space-parallel simulations. However, these categories have a strong impact on the simulation internals, that is, on the simulation algorithm used to compute the model trajectory in parallel.

In this paper, we present a hybrid solution, that combines both space- and time-parallel decompositions, but our aim is to propose a solution that can be implemented using an existing sequential simulator without any change to its implementation. In particular, we do not present an optimization of the time- or space-parallel simulation algorithms, as many examples can be found in literature[3, 17, 19, 22, 23].

Indeed, the performance (or scalability) of a distributed simulation is highly dependent on the amount of time spent in the communications between Logical Processes (LPs) *during* the computation. Therefore, following a classical approach in high performance parallel computing, our solution tries to minimize the time lost in communications. Two obvious directions of investigation can be explored to minimize this loss of time, without trying to optimize the simulation algorithm or simplifying the model: Maximizing the overlap between computations and communications, or reducing the amount of inter-LPs communications. The work we present in this paper contributes mostly to the first direction, but it opens interesting perspectives in the second as well.

1.1 Introducing the solution

Our solution is based on a pragmatic approach, that does not apply to all the simulation scenarios, but only to some particular cases, and in particular to the simulation of large peer-to-peer systems, for which we can make the following hypotheses:

(H1) The systems we study are composed of a large number

SIMUTools 2013 March 5–7, 2013, Cannes, France.

of sub-systems or entities (the peers), noted N;

(H2) The simulation of these sub-systems can be split in space and assigned to a number of Logical Processes, noted P, running on C computing nodes;

(H3) The system has good locality property, such that over a short period of time, called a time-slot and noted h, each sub-system interacts only with a limited number of the other sub-systems; Or stated differently, it is possible to find a period of time sufficiently short, such that no sub-system has enough time over that period to reach more than a fraction of the other peers; we call *model-span* and note $\alpha(h)$ the maximum number sub-systems that can be reached for a period of duration h;

(H4) Assuming that the sending of a message between any two peers takes some time, and that the only interactions that exist in the model between peers are the messages they exchange, we can find a minimal (non-zero) delay of interaction between any two peers, called the causal latency and defined as the smallest amount of time that must elapse between the sending of a message by a peer to another peer, and its notification on the receiver side. This causal latency is a concept similar to the directional distance introduced by Ayani and Rajaei[3].

The solution we present in this paper consists in dividing the overall simulation both in time and space, into a large number of micro-simulation steps. In the space dimension, the division consists in forming a number of clusters of entities, that are each assigned to a given LP. Entities are groups of peers formed according to their locality properties (eg. connected to the same DSL equipement or access network). In time, each LP simulates in parallel with other LPs the behavior of the cluster of entities it is assigned for similar periods of time. The duration of this time period, called a time-step, is chosen such that no communication is needed between LPs until the completion of the time-step: It is chosen sufficiently small to ensure that *no* communication occuring in the simulation model between two peers not belonging to the same LP can be completed (fully simulated) during the same simulation step.

This can be achieved by chosing the duration of the micro-step starting at the simulated time t on LP_i, noted $\theta(t,i)$, such that:

$$\forall t, \forall i \quad \theta(t,i) < L_{min}.$$

where L_{min} is minimal causal latency between two peers that are simulated within two different LPs.

Going further with this idea, we suggest that the duration of the time-step should even be chosen small enough to ensure that the communication inititiated during the time-step i should not complete before the time-step $i+k, k>1$. This way, we enforce a *strong overlap* between the computation of the time-steps and the transmission of their resulting communications. For example, it is easy to show that by choosing $\theta(t,i) < \frac{1}{2}L_{min}$ we can ensure that no communication initiated during time-step i will complete before the end of the time-step $i+1$, which allows to overlap the communications generated by step i with the computations of step $i+1$.

Some level of synchronization is required among LPs, but with some flexibility: a given LP cannot be ahead of the others more than a few steps (ie. no more than the minimum causal latency). The main problem to be solved when reusing an existing sequential simulator in such a time-parallel approach, is the dependency between consecutive time-steps. In our case this dependency mostly consists in passing the workload status of each access network from one time-step to the next one, such that the initial workload of the network at the beginning of a time-step is well taken into account in the computation of the communication times.

1.2 Rationale and observations

The rationale for this solution is that in a non time-stepped parallel discrete-event simulation (PDES), the communications occuring between the LPs at the runtime level are directly related to those occuring in the model between peers: As shown on Figure 1, the simulation of a communication between two peers P_i and P_j may in turn require a communication between two LPs at the runtime level if P_i and P_j happen to be simulated by different LPs.

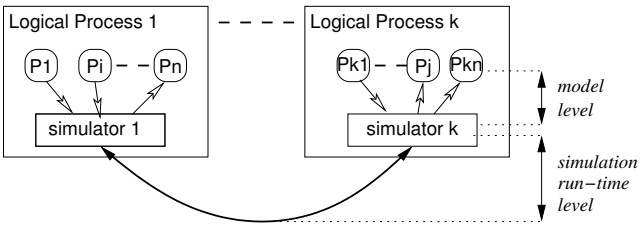

Figure 1: The two levels of communication in a PDES of a Peer-to-peer system model.

Our hypotheses make sense with respect to the typical parameter values we are considering: our goal is to simulate several millions of peers (eg. $N = 10^7$) using a grid computing facility offering up to a few thousands of computing nodes ($C > 10^3$). The number of LP per core also defines the number of peer per LP, and can therefore be used as an adjustement variable to optimize the overall performance of the simulation (this is left as a perspective for further studies). For example, with $P = C$, we end up with an average of 10^4 peers per LP, but this number can be as low as a hundred peers per LP if we chose to have $P = 100 \times C$ (that is a hundred of LPs per computing node), which is not unreasonable to consider on a massively multi-core architecture.

The time-step duration h is in the order of a few tens of milliseconds: If we except the particular case of peers running on the same local network, the typical value of L_{min} is the communication delay observed between any two peers connected to internet through a DSL connexion, which is in the order of a hundred milliseconds. Indeed, if we arrange for peers that are in very close range in the real system to be simulated by the same LP, then the only communications left between LPs are those corresponding to messages exchanged between peers that are not in close range, which leads to the corresponding value of $h = \frac{1}{k}L_{min}$ if we want to achieve a strong overlap.

In hypothesis (H3), the model-span $\alpha(T)$ may be bounded assuming that a communication occuring between any two peers in the system keeps both of them busy and unable to send or receive another message for a minimum amount of

time, noted τ, and such that $\tau > 0$. Hence, the maximum of peers that can be reached over the period of time h of a time-step is such that $\alpha(T) \leq \frac{h}{\tau}$.

This hypothesis makes sense with respect to the previous figures: in a real large-scale peer-to-peer system, even in the most extreme case, the number of peers that can be reached by a single peer in a time interval of a few tens of milliseconds is limited and certainly much smaller than the total number of peers in the system. This limitation is further reinforced by the fact that in a large system, a peer has only a partial view of the system (defined as its neighborhood in the peer-to-peer routing scheme).

Obviously, since our solution makes it impossible for a long-distance full communication to complete during a micro-simulation step, it also discards the corresponding need for communications between LPs at runtime level during the simulation. In addition, since no coordination is required between LPs *during* the micro-simulation steps, most existing sequential event-driven simulation engines can be reused for implementing each simulation micro-step.

However, communications are still needed in order to coordinate the time-steps: Each simulation step starts with a number of pending receptions coming from previous steps, and produces a number of outgoing communications that will be received during a future step.

We also assume that if two peers are sufficiently close (eg. on the same subnet), they should be simulated within the same LP. The dual of this assumption is that any two peers that are simulated by two different LPs are always assumed *not* to be directly contending with each other on a local network. This way, we can safely assume that micro-simulations run in two distinct LPs have no local interactions and can only influence each other's behavior through long distance message passing.

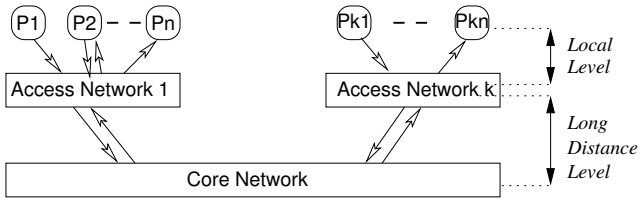

Figure 2: The two levels of network model in a typical Peer-to-peer architecture.

This assumption is consistent with the kind of system we target: Peer-to-peer systems are mostly running in residential networks and connected to the Internet through a DSL connection (or 3G/4G connection in the case of handheld devices) usually refered to as the Access Network. As shown on Figure 2, these local access networks are then connected through a backbone network that we call the Core Network. This is also consistent with the fact that in many simulators, given its complexity, this core network cannot be simulated using the same level of details as the local access network (discovering its actual topology is already a challenging research problem[15]). For example, the contention events can possibly be modeled in detail at the local level, while a more abstract stochastic model seems to be hardly avoidable at the core level.

The actual similarity between Figures 1 and 2 is not fortuitous: NetStep is designed to ensure that within a given LP, the communications occuring between local peers is fully simulated by the (sequential) simulation engine running in the corresponding LP; On the contrary, a communication occuring between two peers not located in the same LP is a long distance communication, and its simulation follows a three phases protocol: (1) simulation of the local communication in the Access Network on the sender side, (2) simulation of Core Network transmission, and (3) simulation of the local communication in the Access Network on the receiver side.

The local communication in the Access Network is handled by the simulator reused for the micro-simulations. For the Core Network transmission, two options can be considered: either rely on the legacy model of the reused simulator, or delegate this part of the simulation to NetStep. In the first case, the computation has to be split between both ends or done fully by one end. In NetStep, the core simulation is done by the source end, such that the time of arrival at the other end of the core network is known. This last option also helps avoiding a performance issue when simulating flooding scenarios: If the simulation was to be done on the receiver side, the receiver LP would have to process all the messages sent to the flooded Peers, which would result in a flooding of the receiver side LP and its simulation engine as well. On the contrary, simulating the core network on the sender side opens interesting perspectives in terms of optimization.

The remainder of this paper is organized as follows: The related works are presented in Section 2; We give a formal definition of the micro-simulation in Section 3, and a description of the NetStep architecture in Section 4; We present our conclusions in Section 5.

2. RELATED WORK

Parallel time-stepped simulation is also commonly called time-parallel simulation, because it divides the simulation model in the time dimension and attempts to process the corresponding divisions in parallel, as opposed to the space-parallel simulation that divides the simulation model in the space dimension[6].

For network simulation, the time-stepped approach is often used to reduce the number of events, as shown for example by Lu and Schormans[12]. They try to reach a reasonable trade-off between loss of accuracy (compared to event driven) and increased computation efficiency. Here, we can distinguish two main approaches: Either trying to reduce the number of events by replacing the packet-related events by some higher level abstraction events, such as packet trains[1] or *chunks* of packets[9], or by replacing the packet model by a fluid model as initially proposed by Kesidis et al. for the simulation of ATM networks[7]. In the latter case, streams of packet are modeled as fluids, and events are associated to the change of fluid levels according to the traffic variations, meaning that the simulator can still be purely event-driven. Yan and Gong showed that a time-driven stepped approach can be used with fluid models and that by varying the duration of the time-step, it allows to change the trade-off between accuracy of the results and complexity of the computation[24]. Although this technique allows better simulation speed-ups, this improvement comes at the cost of an additional tuning step to find the best trade-off.

A closely related approach, proposed by Ayani and Berkman, consists in trying to align the occurrence of events in order to obtain a synchronous processing. This technique,

called time-synchronous parallel simulation, has been shown to be well suited for the SIMD model of parallel execution[2]. Even though the error generated by this forced synchronization can be minimized[17], our approach in NetStep is to avoid model transformation and keep the asynchronous behavior of the model. However, we still have to solve an issue when the initial workload status at the beginning of a time step is missing.

Our solution is actually inspired by the conservative approach proposed by Lubachevsky[13] that consists in defining a period of time, called bounded lag, during which it is safe to process events in parallel. Following a very similar idea, Nicol[14] assigns a global ceiling to each LP. Similarly, Steinman[20] suggests using the notion of Event Horizon to define a safe time period during which events can be processed in parallel. Even though the safe window defined by the event horizon could potentially be larger than the one defined by using the fixed causal latency, as proposed in our algorithm, it comes at the cost of an additional synchronization between the LPs, in order for each to compute and distribute the minimal event horizon among all the LPs.

Furthermore, given the figures we have in mind (several thousands of peers are to be simulated by each LP), we believe there is a high probability that this distributed minimum computation ends up with the same causal latency we already have chosen in our algorithm: It is sufficient that one peer sends a message at the very beginning of the time slot, and that this message needs only the minimal latency to reach a peer simulated by another LP.

Nevertheless, similar to the solution described by Steinman, our solution can benefit from the fact that the events being sent to other LPs will be inserted in the destination event queue at a later time than the time of an equivalent sequential simulation. As shown by Steinman, late insertion may significantly lower the load on the Future Event Set and finally impact positively the performance of the native simulator.

NetStep also has similarities and some shared motivations with the GENESIS tool developed by Szymanski et al.[21]. Indeed both tools divide the simulation model both in time and space, and both aim at reusing existing simulation tools. However, genesis is a more general purpose tool, aimed at the simulation of any large-scale network while NetStep is more specifically geared at the simulation of P2P networks. This difference allows NetStep to make opportunistic assumptions and avoid synchronisation, while GENESIS uses an iterative approach that needs the same time-step to be computed several times until some convergence criterion is met that allows the simulation to proceed with the next time step.

In their Aurora system[16], Park and Fujimoto use a Master/Worker architecture to distribute the simulation work, divided into work units, to computing nodes. Although there are some similarities with our work, their solution requires synchronization algorithms in order to compute lower bound on time-stamp (LBTS), while our solution avoids this requirement by putting restrictions on the kind of models supported, which in turns allows us to reuse in parallel multiple instances of an existing sequential simulator.

Some general (non simulation related) parallelization techniques exist to improve the overlapping between computation and communications. The wait-by-necessity approach offered in GCM[10] is a seamless way of improving the overlapping by using a programming language run-time construction. Our solution deals with the overlapping issue at a higher level by removing the need for synchronous communications during the computation steps. Nevertheless, synchronization communication still occur, but only during the transitions between the computation tasks.

A number of frameworks dedicated to P2P systems engineering have also emerged in the last few years. Splay, for example, provides a lightweight run-time for emulating the behavior of distributed algorithms, including P2P networks[11]. The behavior of the Peers (or agents) is described using a high-level prototyping language, namely Lua. The benefits of using Lua and its extensions for Splay, is to make the code of each agent overly compact. For example, a Chord P2P agent requires no more than a hundred lines of Lua. Splay also provides a distributed run-time with strong isolation and web-services for real-time observations. Compared to Splay, our solution does not try to simplify the run-time or provide a lightweight solution, but rather to scale-up an existing sequential simulator and transform it into a large-scale distributed simulator, with the obvious benefit of being able to reuse all the existing models contributed to that sequential simulator.

3. SIMULATION MICRO-STEPS

A NetStep parallel simulation is divided in many sequential sub-simulations, each of which consisting of a number of micro-steps: The sequential simulations are used to divide the model in the space dimension, while the steps are used to divide each sequential simulation in the time dimension.

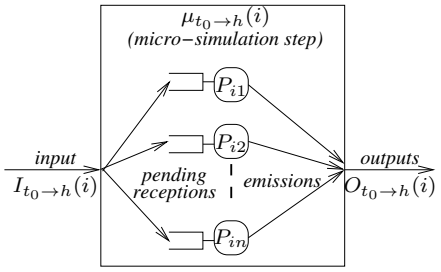

Figure 3: A simulation micro-step

In the spatial dimension, the system under study, which is composed of N entities (Peer, agents or any communicating devices using message passing as a means for interaction), is divided into P sub-systems, which represent groups of clusters of the previous entities. In the time dimension, each of these group is simulated in parallel, for the short period of time corresponding to a time-step, using an unmodified sequential simulator. Each unit of simulation is called a simulation micro-step and is executed by a Logical Process (LP). (A new LP *may* be created for each micro-simulation.) A schematic representation of a simulation micro-step is given Figure 3 (the notations used in the figure are explained hereafter).

We note $\mu_{t_0 \to h}(i)$ the simulation micro-step starting at simulation time t_0 on $LP_i, i \in \{1 \dots P\}$ for a duration $h = \theta(t_0, i)$; this micro-step simulates a sub-model of the system noted $model(i, t_0)$. We note $\mathcal{P}(t_0, i)$ the set of peers being simulated by $model(i, t_0)$ in the micro-simulation step $\mu_{t_0 \to h}(i)$. The set of pending receptions at time t to be com-

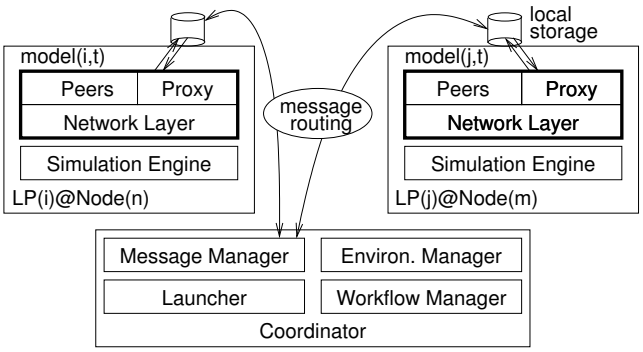

Figure 4: NetStep's architecture

pleted by a peer $P_k \in \mathcal{P}(t_0, i)$ during the step $\mu_{t_0 \to h}(i)$ is noted $R(t, i, k)$, and the set of messages sent from a peer $P_l \in \mathcal{P}(t_0, i)$ in the simulation time interval $[t_0, t[$ of the step $\mu_{t_0 \to h}(i)$, with $t < t_0 + h$, is noted $S(t, i, l)$.

The set of all pending receptions at the begining of the time step $\mu_{t_0 \to h}(i)$ is called the time step *input*, and is noted $I_{t_0 \to h}(i)$:

$$I_{t_0 \to h}(i) = \bigcup_{k \in \mathcal{P}(t_0, i)} R(t, i, k), t \in [t_0; t_0 + h[$$

Similarly, the set of pending messages sent during the time step $\mu_{t_0 \to d}(i)$ is called the time step *output*, and is noted $O_{t_0 \to h}(i)$:

$$O_{t_0 \to h}(i) = \bigcup_{l \in \mathcal{P}(t_0, i)} S(t, i, l), t \in [t_0; t_0 + h[$$

4. ARCHITECTURE

The NetStep architecture consists of a number of Logical Processes (LPs), each executing an instance of the SimGrid simulator, and a Coordinator process.

The Coordinator includes four modules to manage the simulation micro-steps: a work-flow manager, an environment manager, a launcher and a message manager. The work-flow manager dispatches the micro-simulations steps and the corresponding inputs (incoming communications and initial state) to LPs. The environment manager then creates an opportune directory (sandbox) with all the necessary files. The launcher starts the simulation processes. When the peer network topology is fixed, the same simulation process may be used several times to run multiple time-steps in sequence; Otherwise, a new simulation process has to be started when the network topology varies between time-steps. The last module, the message manager, uses a client/server pattern to collect the un-dispatched messages from the various LPs. It memorizes them temporarily, so that they can reach the destination in the next simulation micro-step. If the number of the messages is too high for the available memory, a distributed message manager could be easily implemented.

In order to run a simulation micro-step without modifying the simulation engine, NetStep introduces a virtual simulation entity in each LP. This virtual entity acts as a proxy for the communications with external entities (peers located in other LPs): On their way out, the departing messages first go through the sender's access network and get routed to the proxy through a link that simulates the core network. Given the delay due to the core network, the message is expected after the end of the micro-simulation step and gets stored by the proxy as part of the micro-step's outputs. After the end of the micro-step, the coordinator routes the outputs of the last step to the corresponding future steps where they become the new inputs. It is important to note here that the future step is not necessarily the one coming right after the last one but possibly several steps later, depending on the core network delays. When the simulation running the step of the time of arrival starts, the pending inputs for that step are read by the LP proxy that schedules them to be routed on their expected time of arrival to the destinaton peer through the receiver's access network.

5. CONCLUSION

In this paper we presented NetStep, a framework that helps to distribute a simulation, dividing the execution both in spatial and temporal dimensions, using multiple LPs running in parallel series of simulation micro-steps. The time-stepped simulation is used to optimize the simulation of the entities and of the events coordinating the simulation of small time windowed simulation steps. We implemented our solution by reusing the SimGrid simulation engine in order to reuse existing models, and tested on a simple peer-to-peer model.

Our main motivation for this work is to be able to simulate very large scale *border* networks such as peer-to-peer networks or networks of handheld devices by reusing existing sequential simulators.

However, as mentioned in introduction, since our solution allows for a loose-synchronization between consecutive time-steps, some data may be missing at the beginning of a time-step, such as the workload status of the access network at the end of the previous step. We have started working on various solutions to this problem. For example, assuming that the only effect of the workload is to increase the communication delay in the source and destination access network, we first compute a slightly erroneous schedule for the sending and receiving events by assuming no workload, and then we make a correction to the schedule once the initial workload level is received from the previous time-step. This solution seems to solve well the case of outgoing communications, that can easily be buffered until the workload level is finally received (possibly as late as the end of the time-step); however this correction may come too late for the early receptions and produce a slightly erroneous (optimistic) reception schedule. Another idea to be investigated in the case of early receptions, is to use the last available workload level (possibly coming from a few steps back in the past), instead of waiting for the very last one. This is not fully accurate, but we think it might be safe to assume that the workload variations over reasonably short periods of time are sufficiently small to make this approximation acceptable.

It should also be noted that our solution only addresses the parallel execution of the simulation model, which leaves aside the parallel execution of other common tasks found in a simulation, such as the on-line statistics computation. Parallelizing all these activities is a challenging issue as long as those activities are coded inline with model. However, following our previous works[18], we strongly believe that these problems can be dealt separately by using proper separation of concerns techniques.

6. ACKNOWLEDGEMENTS

This work is partly funded by the SONGS project of the French National Research Agency, under contract ANR-11-INFRA-13.

7. REFERENCES

[1] Jong Suk Ahn and Peter B. Danzig. Packet network simulation: Speedup and accuracy versus timing granularity. *IEEE/ACM Transactions on Networking*, 4(5):743–757, October 1996.

[2] R. Ayani and B. Berkman. Parallel discrete event simulation on SIMD computers. *Journal of Parallel and Distributed Computing*, 18(4):501–508, 1993.

[3] Rassul Ayani and Hassan Rajei. Parallel simulation using conservative time window. In *Proc. of the 1992 Winter Simulation Conference*, WSC'92, pages 709–717, 1992.

[4] K.M. Chandy and R. Sherman. Space, time, and simulation. In *Proc. of the SCS Multiconference on Distributed Simulation*, SCS Simulation Series, 1989.

[5] O. Dalle, F. Giroire, J. Monteiro, and S. Perennes. Analysis of failure correlation impact on peer-to-peer storage systems. In *Proc. of the 9th IEEE Intl. Conf. on Peer-to-Peer Computing*, P2P '09, pages 184 –193, sept. 2009.

[6] R. M. Fujimoto. *Parallel and Distributed Simulation Systems*. Wiley Interscience, 2000.

[7] D. Cheung G. Kesidis, A. Singh and W.W. Kwok. Feasibility of fluid-driven simulation for ATM network. In *Proceedings of IEEE GLOBECOM*, volume 3, pages 2013–2017, November 1996.

[8] Z. Ge, D.R. Figueiredo, Sharad Jaiswal, J. Kurose, and D. Towsley. Modeling peer-peer file sharing systems. In *Proc. of the 22d Annual Joint Conf. of the IEEE Computer and Communications Societies*, volume 3 of *INFOCOM 2003*, pages 2188 – 2198 vol.3, march-3 april 2003.

[9] Yang Guo, Weibo Gong, and D. Towsley. Time-stepped hybrid simulation (TSHS) for large scale networks. In *Proc. of the 9th Annual Joint Conf. of the IEEE Computer and Communications Societies*, volume 2 of *INFOCOM 2000*, pages 441–450 vol.2, 2000.

[10] Ludovic Henrio, Florian Kammüller, and Marcela Rivera. An asynchronous distributed component model and its semantics. In Frank S. de Boer, Marcello M. Bonsangue, and Eric Madelaine, editors, *Formal Methods for Components and Objects, 7th International Symposium, FMCO 2008, Sophia Antipolis, France, October 21-23, 2008, Revised Lectures*, volume 5751, pages 159 – 179. Springer, 2009.

[11] L. Leonini, E. Riviere, and P. Felber. P2P experimentations with Splay: From idea to deployment results in 30 min. In *Proc. of the 8th IEEE Intl. Conf. on Peer-to-Peer Computing*, P2P '08, pages 189–190, sept. 2008.

[12] S. Lu and J.A. Schormans. Time-stepped approach for accelerated simulation of mobile ad hoc networks. *Communications, IET*, 2(5):609–620, may 2008.

[13] Boris D. Lubachevsky. Bounded lag distributed discrete event simulation. In *Proc. of the SCS Western Multiconference on Distributed Simulation*, pages 183–191, 1988.

[14] David Nicol. Performance bounds on parallel self-initiating discrete-event simulation. *ACM Trans. on Modeling and Computer Simulation*, 1(1):24–50, 1991.

[15] Ricardo V. Oliveira, Dan Pei, Walter Willinger, Beichuan Zhang, and Lixia Zhang. In search of the elusive ground truth: the internet's as-level connectivity structure. In *Proc. of the 2008 ACM SIGMETRICS Intl. Conf. on Measurement and Modeling of Computer Systems*, SIGMETRICS '08, pages 217–228, New York, NY, USA, 2008. ACM.

[16] Alfred Park and Richard M. Fujimoto. Aurora: An approach to high throughput parallel simulation. In *Proc. of the 20th Workshop on Principles of Advanced and Distributed Simulation*, PADS '06, pages 3–10, Washington, DC, USA, 2006. IEEE Computer Society.

[17] Hyungwook Park and Paul A. Fishwick. A fast hybrid time-synchronous/event appraach to parallel discrete event simulation of queuing networks. In *Proc. of the 2008 Winter Simulation Conference*, WSC '08, pages 795–803, December 2008.

[18] Judicaël Ribault, Olivier Dalle, Denis Conan, and Sébastien Leriche. OSIF: A Framework To instrument, Validate, and Analyze Simulations. In *Proc. of 3rd Intl. ICST Conference on Simulation Tools and Techniques*, SIMUTools'2010, Torremolinos, Spain, 15âĂŞ19 March 2010.

[19] George F. Riley, Talal Mohamed Jaafar, Richard M. Fujimoto, and Mostafa H. Ammar. SpaceâĂŞparallel network simulations using ghosts. In *Proc. of the 18th Workshop on Principles of Advanced and Distributed Simulation*, PADS '04, pages 170 – 177, May 2004.

[20] Jeff S. Steinman. Discrete-event simulation and the event horizon. In *Proc. of the 8th Workshop on Parallel and Distributed Simulation*, PADS '94, pages 39–49, New York, NY, USA, 1994. ACM.

[21] Boleslaw K. Szymanski, Adnan Saifee, Anand Sastry, Yu Liu, and Kiran Madnani. Genesis: A system for large-scale parallel network simulation. In *Proc. of the 16th Workshop on Parallel and Distributed Simulation*, PADS'02, pages 89–96. IEEE CS Press, 2002.

[22] Jain J. Wang and Marc Abrams. Massively time-parallel, approximate simulation of loss queueing systems. *Annals of Operations Research*, 53(1):553–575, 1994.

[23] H. Wu, R.M. Fujimoto, and M. Ammar. Time-parallel trace-driven simulation of CSMA/CD. In *Proc. of the 7th Workshop on Parallel and Distributed Simulation*, PADS '03, pages 105 – 112, june 2003.

[24] Anlu Yan and Wei-Bo Gong. Time-driven fluid simulation for high-speed networks with flow-based routing. *IEEE Transactions on Information Theory*, 45(5):1588–1599, July 1999.