2011 IEEE Workshop on Principles of Advanced and Distributed Simulation

25th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2011)

14 - 17 June 2011
Nice, France

IEEE Catalog Number: CFP11038-USB
ISBN: 978-1-4577-1365-1
ISSN: 1087-4097

2011 IEEE Workshop on Principles of Advanced and Distributed Simulation

IEEE Catalog Number: CFP11038-USB
ISBN: 978-1-4577-1365-1
ISSN: 1087-4097

Printed copies of this publication are available from:

Curran Associates, Inc
57 Morehouse Lane
Red Hook, NY 12571 USA
Phone: (845) 758-0400
Fax: (845) 758-2633
E-mail: curran@proceedings.com

Produced by the IEEE eXpress Conference Publishing
For information on producing a conference proceedings and
quotations, contact conferencepublishing@ieee.org
http://www.ieee.org/web/publications/pubservices/confpub/index.html

Message from the General Chair

Bienvenue à PADS 2011.

It is my great pleasure and honor to organize this year's edition of the PADS Workshop in Nice.

Besides some of the well-known wonders that attract many visitors to this region -- our local cuisine and Mediterranean beaches among them -- Nice is also home to some very significant research activity, both at the Science Campus of the University of Nice Sophia Antipolis and in the neighboring Technopole of Sophia Antipolis -- occasionally known as the French "Silicon Valley." This area and its new ICST Campus hosts a number of internationally reknown research labs, including INRIA, CNRS, INRA and Eurecom, as well as leading hi-tech industries.

I hope the proximity of these to this year's PADS Workshop will allow many of you to establish fruitful contacts here, now and in the future, while allowing my local colleagues to discover and enjoy the quality of the research that has contributed to the PADS Workshop's exceptional reputation. Going one step further, I think that we, simulation specialists, should ease access to this exciting research field for new comers and emphasize its research potential and how so many open issues still need to be addressed.

PADS is a well known venue where so many of these issues have been addressed and solved. I have no doubt this year's contributions will, once more, pave the way towards better, bigger and faster simulations!

I look forward to the positive exchange of great ideas and wish you all an excellent PADS workshop and a very pleasant stay in Nice.

Olivier Dalle,
PADS 2011 General Chair

Message from the Program Chair

A warm welcome to PADS 2011, the 25th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation. This year's conference has attracted a significant number of high quality submissions - a fact that we not only attribute to the fact of having the stunning Nice (France) as our conference site, but also to the past years' efforts of the steering committee to broaden the scope of and the participation in the conference.

Applying the traditional and rigorous double blind review process of PADS we have ensured a very high quality of papers which made it into the final program. From the original number of 61 submitted abstracts 48 full papers were submitted by the deadline and entered the review process. From these 48 papers, 22 papers were accepted to the conference, resulting in an acceptance rate of 46 %. From these 22 papers 2 papers were accepted as short papers, and 8 papers were shepherded into the conference – another PADS measure of ensuring high quality contributions.

We are also very pleased to have Simon J E Taylor from Brunel University (UK) and David R.C. Hill from Blaise Pascal University (France) as our two keynote speakers.

In staying with the PADS tradition, a panel of program committee members will select the best paper for the PADS 2011 Best Paper Award from the papers ranked most highly by the reviewers. The nominees for this year's Best Paper Award are (in no particular order):

- **A Virtual Time System for OpenVZ-Based Network Emulations**
  Yuhao Zheng and David Nicol
- **Parallel Discrete Event N-body Dynamics**
  Matthew Holly and Carl Tropper
- **Efficiently Scheduling Multi-core Guest Virtual Machines on Multi-core Hosts in Network Simulation**
  Srikanth Yoginath and Kalyan Perumalla
- **Application Transparent Migration of Simulation Objects with Generic Memory Layout**
  Sebastiano Peluso, Diego Didona, and Francesco Quaglia

Congratulations to all of the finalists for an excellent set of papers! The winner will be announced at the conference banquet.

On behalf of the PADS 2010 program co-chairs, I am further pleased to report that the PADS 2010 Best Paper Award went to "Ad Hoc Distributed Simulation of Queueing Networks", by Ya-Lin Huang, Christos Alexopoulos, Michael Hunter, and Richard Fujimoto.

Finally, I would like to use the chance to thank all authors, program committee members and additional reviewers for putting so much time and effort into making this conference a success! A special thank you goes to Francesco Quaglia as one of past year's program co-chairs for helping me on several occasions, and to Olivier Dalle, our general chair, for taking care of so many organizational details that made my life as program chair so much easier.

I hope you enjoy the conference and the program!

Steffen Strassburger
Ilmenau University of Technology, Germany

Program Committee

R. Ayani	Royal Institute of Technology, Sweden
S. Balsamo	Univ. of Venezia, Italy
W. Cai	Nanyang Tech. Univ., Singapore
C. Carothers	Rensselaer Polytech. Inst., USA
L. Donatiello	Univ. of Bologna, Italy
R. M. Fujimoto	Georgia Inst. of Tech., USA
D. Jefferson	Lawrence Livermore Nat. Lab, USA
M. Hybinette	Univ. of Georgia, USA
C. Laroque	Univ. of Paderborn, Germany
M. Liljenstam	Ericsson Research, Sweden
J. Liu	Florida International Univ., USA
D. M. Nicol	Univ. of Illinois, USA
K. S. Perumalla	Oak Ridge National Lab, USA
P. F. Reynolds	Univ. of Virginia, USA
G. F. Riley	Georgia Inst. of Tech., USA
T. Schulze	Univ. of Magdeburg, Germany
S. J. E. Taylor	Brunel Univ., UK
A. Tolk	Old Dominion University, USA
Y. M. Teo	National Univ. of Singapore
C. Tropper	McGill Univ., Canada
S. J. Turner	Nanyang Tech. Univ., Singapore
A. Uhrmacher	Univ. of Rostock, Germany
F. Quaglia	Univ. of Rome, Italy
A. Verbraeck	TU Delft, The Netherlands
P. A. Wilsey	Univ. of Cincinnati, USA

Additional Reviewers:

Luciano Bononi, Gabriele D'Angelo, Marco Di Felice, Roland Ewald, Jan Himmelspach, Ya-Lin Huang, Daniel Huber, Michael Lees, Stefan Leye, Imran Mahmood, Alke Martens, Moreno Marzolla, Katharina Rasch, Cole Sherer, Steve Smith, Claudia Szabo, Marco Vassura, George Vulov, Irfan Younas.

Steering Committee

S. J. Turner (Chair)	Nanyang Tech. Univ., Singapore
A. S. Elmaghraby (IEEE)	Univ. of Louisville, USA
J. Liu	Florida International Univ., USA
D. Jefferson	Lawrence Livermore National Laboratories, USA
F. Quaglia	Univ. of Rome, Italy
G. F. Riley (ACM)	Georgia Inst. of Tech., USA
B. W. Unger (SCS)	Univ. of Calgary, Canada
R. M. Fujimoto (Honorary)	Georgia Inst. of Tech., USA

Table of Contents

Session 6: GPU and Multicore Computing

Session 7: Simulation Techniques

Realising Parallel and Distributed Simulation in Industry: A Roadmap

Simon J E Taylor
ICT Innovation Group, Dept of Information Systems and Computing
Brunel University
Uxbridge, UK
simon.taylor@brunel.ac.uk

***Abstract*—Modeling & Simulation practitioners use Commercial-off-the-shelf Simulation Packages to model process-based system problems in a wide range of domains. Technological limitations of these packages restrict what practitioners can model in terms of simulation run times and system boundaries. Experiences in dealing these problems with Grid Computing and Distributed Simulation are presented and a roadmap of challenges that need to be met in order for practitioners to overcome these limitations is developed.**

Modeling and Simulation Commercial-off-the-shelf Simulation Packages; Distributed Simulation; Grid Computing; Industry

I. OVERVIEW

Modeling & Simulation (M&S) is widely used in industry to analyze process-based production and logistics problems across manufacturing and defense, and related problems in traffic planning and supply chains. It is also used to study clinical pathways and resource management in healthcare. Commercial-off-the-shelf Simulation Packages (CSPs) are visual interactive modeling environments that support the development, experimentation and visualization of simulation models. These support different M&S world views such as real-time simulation, discrete-event simulation, agent-based simulation and system dynamics. Of these, arguably, discrete-event simulation is the most widely used in industry.

The contemporary practice of discrete-event simulation is facing significant and practical barriers in terms of time and system boundary. Time limits the amount of experimentation can be performed within a simulation project and leads to either model depth being sacrificed or investigation left incomplete. System boundary limits the scope of a simulation to what can be practically modeled within a single CSP and prevents M&S practitioners from addressing large scale industrial problems. Distributed computing, and in particular Parallel and Distributed Simulation, can address these issues.

Experiences with dealing with time and system boundary M&S problems in industry through the application of Grid computing and Distributed Simulation are presented [1-3]. It argues that effective and careful collaboration between end user practitioners, vendors and researchers is key and that researchers play a vital role in the successful knowledge transfer of these techniques from academia to industry. The role of standardization is also considered with recent experiences in developing SISO-STD-006-2010 *Commercial-off-the-shelf Simulation Package Interoperability Reference Models* [4] and collaboration with the European Desktop Grid Initiative. A roadmap of challenges that need to be met to allow M&S practitioners to break through these barriers to reduce project costs, perform more effective investigation and to study larger systems is developed.

REFERENCES

[1] S. J. E. Taylor, S. J. Turner and S. Strassburger Guidelines for Commercial-Off-The-Shelf Interoperability. In *Proceedings of the 40th Winter Simulation Conference*, 2008, pp. 193-204. ACM Press, New York, NY.

[2] S. J. E. Taylor, S. J. Turner and S. Strassburger, N. Mustafee and K. Pan. Commercial-Off-The-Shelf Simulation Package Interoperability: Issues and Futures. In *Proceedings of the 41st Winter Simulation Conference*, 2009 pp. 203-215. ACM Press, New York, NY.

[3] S. J. E. Taylor, Mustafee, N., Kite, S., Wood, C., S. Strassburger and S. J. Turner. Improving Modeling and Simulation Through Advanced Computing Techniques: Grid Computing and Distributed Simulation. In Proceedings of the 42nd Winter Simulation Conference, 2010, pp. 216-230. ACM Press, New York, NY.

[4] SISO-STD-006-2010 Standard for COTS Simulation Package Interoperability Reference Models. 2010. Simulation Interoperability Standards Organization, Orlando, Florida.

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

Distribution of Random Streams in Stochastic Models in the age of multi-core and manycore processors

David R.C. Hill
ISIMA/LIMOS – UMR CNRS 6158 – Blaise Pascal University
Clermont Université – Campus des Cézeaux – BP 10125
63173 AUBIERE CEDEX
FRANCE
David.Hill@univ-bpclermont.fr

Abstract—**Stochastic Modeling & Simulation require very good random sources which have now been available for more than a decade. However the parallelization of random streams remains delicate for many practitioners. Recent experience shows that a misuse of pseudo-random number parallelization techniques is not infrequent in various simulation domains. This talk will present the state of the art in the distribution of pseudo-random numbers and will also discuss the use of such techniques for hybrid computing with GP-GPUs.**

Random Numbers, Parallelization of random streams, Stochastic Modeling, Distributed Simulation; GP-GPU computing

I. OVERVIEW

Random number generators are necessary in every simulation which includes stochastic aspects. For High Performance Computing, there is an increasing interest in the distribution of parallel random number streams. Even if we have now at our disposal statistically sound random number generators, according to very tough testing libraries [5] [6], their parallelization can still be a delicate problem [7]. A set of recent publications shows it still has to be mastered by the scientific community [1].

In this presentation, we discuss the different partitioning techniques currently in use to provide independent streams with their corresponding software. A state of the art in the parallelization of random numbers for High Performance Computing is given from the point of view of a simulation practitioner. With the arrival of multi-cores and manycore processors architecture on the scientist's desktop, modelers who are non-specialists in parallelizing stochastic simulations need help and advice in distributing rigorously their experimental plans and replications according to the state of the art in pseudo-random numbers partitioning techniques [3].

New programming models and platforms are now available for a larger audience, particularly with the arrival of very powerful GP-GPUs (General Purpose Graphical Processing Units). In addition to the classical approaches in use to parallelize stochastic simulations on regular processors, this talk will also present the recent advances in pseudo-random number generation for GP-GPUs with their associated parallelization techniques [2] [4].

REFERENCES

[1] D. Hill, "Practical distribution of random streams for stochastic high performance computing," in IEEE International Conference on High Performance Computing & Simulation, 2010, pp. 1–8.

[2] J. Passerat-Palmbach, C. Mazel, A. Mahul, and D. Hill, "Pseudo-Random Number Generation on GP-GPU", in Proceedings of PADS : Principles of Advanced and Distributed Simulation, 2011, to be published.

[3] J. Passerat-Palmbach, C. Mazel, A. Mahul, and D. Hill, "Reliable initialization of gpu-enabled parallel stochastic simulations using Mersenne Twister for Graphics Processors," in Proceedings of the European Simulation Symposium, 2010, pp. 187–195.

[4] M. Saito, M. Matsumoto, "A variant of Mersenne Twister suitable for Graphics Processors," http://arxiv.org/abs/1005.4973, 2010, submitted.

[5] P. L'Ecuyer, ``Pseudorandom Number Generators", in Encyclopedia of Quantitative Finance, R. Cont, Ed., in volume Simulation Methods in Financial Engineering, E. Platen and P. Jaeckel Eds., John Wiley, Chichester, UK, 2010, pp. 1431-1437.

[6] P. L'Ecuyer and R. Simard, "Testu01: A C library for empirical testing of random number generators," ACM Transactions on Mathematical Software, Vol. 33, No. 4, August 2007, pp. 22:1–40.

[7] R. Reuillon, D. Hill, Z. El Bitar and V. Breton, "Rigorous distribution of stochastic simulations using the DistMe toolkit", IEEE Transactions On Nuclear Science, Vol. 55, No. 1, 2008, pp. 595-603.

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

Parallel Discrete Event N-body Dynamics

Matthew Holly and Carl Tropper
School of Computer Science
McGill University
Montreal, Canada
matthew.holly@mail.mcgill.ca, carl@cs.mcgill.ca

***Abstract*—Numerical simulation of gravitational N-body systems is an important tool for studying the dynamic behaviour of stellar systems, and in some cases is the only option available given the extremely large time scales involved. The direct summation approach, which evaluates the force between each pair of particles at each time step, produces the most accurate results. However despite many algorithmic advances this method remains a computationally challenging problem owing to its $O(N^2)$ scaling characteristics. The desire to model increasingly larger systems has spurred the adoption of parallel computation techniques, but unfortunately many of the strategies used to accelerate sequential direct N-body simulations hinder their efficient parallelization. This paper investigates the use of parallel discrete event simulation as an alternative to the usual iterative time-stepping approach. By decomposing typical operations into finer-grained events, it is shown that there exists considerable potential for exploiting the model's inherent concurrency. In addition, it is demonstrated how certain optimizations that are normally difficult to parallelize are incorporated naturally into the parallel discrete event paradigm.**

Keywords-parallel, distributed, discrete event, numerical simulation, n-body dynamics.

I. INTRODUCTION

The simulation of physical systems, often modelled by partial differential equations covering multiple spatial and temporal scales, has traditionally been one of the main driving forces behind research in the field of high-performance computing. Larger and more complex models place ever-increasing demands on computational resources, and in many cases harnessing the potential of parallel computers is the only way to achieve significant performance improvements.

Most algorithms for simulating physical systems employ a time-stepping approach. The model is represented by variables and data structures at some moment in simulation time t. In order to integrate the state forward, the simulation clock is advanced by some small amount Δt, and the new state is computed to represent the model at time $t+\Delta t$. The process is thereafter repeated until the simulation clock reaches some maximum value or some other convergence criterion has been met. The parallel implementation of time-stepping algorithms is relatively straightforward. Each CPU is allocated a portion of the model, and each integrates its portion of the global state forward from time t to time $t+\Delta t$, whereupon all processes synchronize and exchange updated state information.

Parallel and distributed discrete event simulation (PDES) is an alternative to the more common time-stepping or time-driven approaches, and has successfully been used to accelerate the execution of spatially discretized physical models [16]. Domain decomposition is performed as before, however afterwards the processes advance their subset of the state variables independently at their own rate, communicating asynchronously via time-stamped messages.

Unlike many time-stepping schemes which force a costly global synchronization after every major integration step, PDES synchronization strategies fall into two broad categories: conservative or optimistic. Conservative mechanisms avoid processing any messages beyond a certain simulation time until it can be ascertained that no message with a lower time stamp can possibly arrive [4], [14]. In other words, conservative methods avoid scenarios where an incoming message could arrive in the logical past, which would indicate a causality violation. Optimistic PDES algorithms, on the other hand, do not wait until it is safe to process a message, but do provide mechanisms to detect and recover from causality violations should they occur [10]. By speculatively executing messages and avoiding the inefficiencies associated with global synchronization, optimistic strategies are ideal for extracting a model's inherent parallelism.

In addition to conservative and optimistic synchronization strategies, it is possible to create application-specific protocols when necessary. A given application may present a particular set of requirements, or may be important enough to merit such customized treatment. We argue that the gravitational N-body problem is just such a problem, and present a parallel event-driven synchronization algorithm that exploits additional concurrency not accessible to either conservative or optimistic methods.

A. Paper outline

This paper is structured as follows. Section II discusses the time-stepping and event-driven simulation methods, with a focus on highlighting the differences between the two strategies. Section III describes the N-body problem, including

a description of the integration scheme employed to advance particle data. This section also details the impediments to efficiently porting optimizations useful for sequential simulations to parallel computers. Section IV builds on the previous two sections, describing a parallel discrete event implementation of an N-body solver. Section V provides an analysis of the performance of the discrete event algorithm. Section VI covers future research and possible extensions of the work presented herein, while section VII concludes.

II. Discrete simulation methods

Simulation methods that discretize time are distinguishable by the manner in which they order and control the flow of simulation time. The most common strategies for advancing simulation time are normally either of the time-stepped or event-driven variety [5].

A. Time-stepped execution

In the time-stepped approach simulation time is advanced in constant-sized steps, with state variables being updated as the system progresses through the sequence of steps. The choice of step size is crucial to both the performance and accuracy of the time-stepping method. Choosing a time step that minimizes computational costs while still managing to capture essential simulation characteristics often requires care and experience.

Time-stepping schemes lend themselves naturally to parallel implementation. Each processor is assigned a subset of the state space and is responsible for updating only that subset. Each CPU is then free to advance its own *local* simulation time and update its own segment of the global state space. The simulation proceeds in a lock-step pattern, with all processors required to block at the end of each step in order to synchronize newly updated state data.

For any given time step it's possible that some processors will have less work to do than others and hence will spend a relatively larger amount of time idling at the global synchronization barrier, unable to continue until the others catch up. The pathological case has only a single heavily-loaded CPU active during a given time step, while all others remain idle. Proper load balancing, arising from the initial partitioning of the state space and potentially dynamically adjusted over the course of the simulation, is crucial to maximizing the performance of the time-stepping schemes.

B. Event-driven execution

The trial-and-error process of choosing the right time step is problematic. Rather than updating the state of the system at fixed intervals, the event-driven approach targets pertinent state changes that take place at instantaneous moments in simulation time. Events scheduled at specific times are processed by invoking application-defined event handling functions.

By focusing on interesting moments over the course of the simulation, event-driven schemes have the potential to be more efficient that their time-stepping counterparts. In particular, when the state trajectory is relatively stable except for a few key irregularly dispersed moments, the event-driven method does not waste time recomputing state information when nothing much is going on.

Between the time-stepping and event-driven paradigms, the latter is traditionally seen as the more complex to parallelize [5], [6]. The first difficulty stems from the absence of a fixed time step known to all processors, such that at any given moment in wall clock time one should expect a disparity amongst the local simulation times. The processors advance asynchronously and it becomes more difficult to know when it is safe to advance local state, as receiving a time-stamped message in one's logical past would indicate that a causality violation has occurred.

III. The N-body Problem

This section provides the background information on the N-body problem necessary to later introduce the parallel discrete event implementation of an N-body solver.

A. Overview

Despite a long history, N-body simulation remains a computationally challenging problem. The general N-body problem refers to the study of the dynamics of N interacting bodies, in particular the motion of particles under the influence of their mutual gravitational attraction. N-body simulations have been used to study a wide range of astrophysical models, from the dynamics of galaxies containing billions of stars to globular clusters containing on the order of 10^6 stars, down to the relatively small scale of planetary systems containing dozens of bodies [1], [3].

An N-body system consists of N particles, represented as infinitesimal point masses. The total force acting on a particle is the sum of its pairwise interactions with all other particles.

$$F \equiv m_i a_i = m_i G \sum_{j=1, j\neq i}^{N} m_j \frac{\boldsymbol{x}_i - \boldsymbol{x}_j}{|\boldsymbol{x}_i - \boldsymbol{x}_j|^3} \qquad (1)$$

B. Numerical Method

The dynamic evolution of particles in the system is driven by their time-varying mutual gravitational field. Direct summation approaches involve the straightforward application of Newton's law (1). The force on each particle is calculated by accumulating the contributions of all other particles in the system, for a total of N(N-1)/2 partial force calculations. By explicitly computing the inter-particle forces for all pairs of particles in the system, this approach scales as $O(N^2)$ and is thus computationally very demanding for systems involving large numbers of particles. Direct evaluation methods are typically employed for star clusters containing on the order of 10^4 stars [12], whereas the use of special-purpose hardware

has enabled systems with 10^5 stars to be studied using direct evaluation techniques [11].

A variety of methods have been developed to alleviate the burdensome $O(N^2)$ scaling properties of the brute force scheme. The basic idea is to reduce the number of direct particle-particle force evaluations by using approximation techniques. For example, the Barnes-Hut tree method works by partitioning the simulation volume into a hierarchical tree of axis-aligned sub-volumes or cells [2]. The leaf nodes of the tree contain the particles, and the force on each particle is calculated by walking the tree. The hierarchical nature of the tree allows for the gravitational potential due to entire groups of particles to be approximated by low-order multipole expansion [2].

Despite the advantageous scaling characteristics of the Barnes-Hut scheme and its ability to handle significantly more particles, in many cases the direct calculation method remains preferable owing to its superior accuracy, algorithmic simplicity, and the fact that no *a priori* assumptions need to be made regarding the system being simulated (e.g. uniform particle distribution) [15]. For these reasons we shall restrict ourselves to direct summation methods for the remainder of this paper.

C. Individual Time Steps

In the physical systems modelled by N-body simulations it is typical to observe various phenomena operating at vastly different time scales [1]. Correspondingly, the optimal choice of time step to best capture particle interactions can vary greatly over the course of each particle's trajectory.

Individual time steps (ITS) are a significant improvement over the constant time step method, offering several orders of magnitude increase in performance as well as improved accuracy [13]. The idea is to assign each particle P_i a suitable time step Δt_i, reserving small time steps for only those particles that require them while allowing others to take larger steps whenever possible. In essence, stepping forward on a per-particle basis is a response to a commonly observed characteristic of N-body simulations wherein few particles require frequent and smaller time steps in order to preserve the numerical stability of their trajectory [1]. Most direct N-body codes incorporate some variation of the individual time step approach, however the use of ITS often precludes an efficient parallel implementation [17].

D. The Hermite Scheme

In this section we describe the widely-used 4th order Hermite integration scheme with individual time steps [12], presented here in its predictor-corrector form. Each particle P_i has mass m_i, position $\mathbf{x}_i$, velocity $\mathbf{v}_i$, acceleration $\mathbf{a}_i$, as well as the first time derivative of the acceleration $\mathbf{k}_i$ (known as the "jerk"). In the case of individual time steps, each particle also has its own time t_i and step Δt_i. A particle's time reflects the last time it was updated, and its update time $ut_i = t_i + \Delta t_i$ is the next time at which the forces acting on the particle will be calculated.

The time-stepping algorithm proceeds iteratively, as follows:

1) Each particle's initial time step is calculated using (2). As per Makino and Aarseth [12], a scaling constant with value of $\eta_s \sim 0.01$ is typically sufficient.

$$\Delta t_i = \eta_s \left| \frac{\boldsymbol{a}_i}{\boldsymbol{k}_i} \right| \tag{2}$$

2) Select particle P_i with the smallest update time $ut_i = t_i + \Delta t_i$ and set the global simulation clock to this time.

3) Predict the positions and velocities of all particles to the update time ut_i. This prediction is done via extrapolation using the values $\mathbf{x}$, $\mathbf{v}$, $\mathbf{a}$ and $\mathbf{k}$, as per (3) and (4). Note that in general for individual time step methods each particle will have its own local time and hence the value of $dt_j = ut_i - t_j$ will differ.

$$\boldsymbol{x}_{pj} = \boldsymbol{x}_j + \boldsymbol{v}_j dt_j + \frac{1}{2}\boldsymbol{a}_j {dt_j}^2 + \frac{1}{6}\boldsymbol{k}_j {dt_j}^3 \tag{3}$$

$$\boldsymbol{v}_{pj} = \boldsymbol{v}_j + \boldsymbol{a}_j dt_j + \frac{1}{2}\boldsymbol{k}_j {dt_j}^2 \tag{4}$$

4) Calculate the acceleration $\mathbf{a}_p$ and jerk $\mathbf{k}_p$ using direct summation as per (6) and (7). This is the most compute-intensive part of any N-body kernel.

$$\begin{aligned} \boldsymbol{r}_{ij} &= \boldsymbol{x}_{pj} - \boldsymbol{x}_{pi} \\ \boldsymbol{v}_{ij} &= \boldsymbol{v}_{pj} - \boldsymbol{v}_{pi} \end{aligned} \tag{5}$$

$$\boldsymbol{a}_p = \sum_j G m_j \frac{\boldsymbol{r}_{ij}}{(r_{ij}^2 + \varepsilon^2)^{3/2}} \tag{6}$$

$$\boldsymbol{k}_p = \sum_j G m_j \frac{\boldsymbol{v}_{ij}}{(\boldsymbol{r}_{ij}^2 + \varepsilon^2)^{3/2}} + \frac{3(\boldsymbol{v}_{ij} \cdot \boldsymbol{r}_{ij})\boldsymbol{r}_{ij}}{(\boldsymbol{r}_{ij}^2 + \varepsilon^2)^{5/2}} \tag{7}$$

5) Using a Hermite interpolation based on the acceleration and jerk, calculate the higher order derivatives of the acceleration with (8) and (9).

$$\ddot{\boldsymbol{a}}_i = \frac{-6(\boldsymbol{a}_i - \boldsymbol{a}_p) - dt(4\boldsymbol{k}_i + 2\boldsymbol{k}_p)}{dt^2} \tag{8}$$

$$\ddot{\boldsymbol{k}}_i = \frac{12(\boldsymbol{a}_i - \boldsymbol{a}_p) + 6dt(\boldsymbol{k}_i + \boldsymbol{k}_p)}{dt^3} \tag{9}$$

6) The predicted position and velocity from step 3 are then corrected to higher order using (10) and (11).

$$\boldsymbol{x}_i = \boldsymbol{x}_p + \frac{\ddot{\boldsymbol{a}}_i}{24} dt^4 + \frac{\ddot{\boldsymbol{k}}_i}{120} dt^5 \tag{10}$$

$$\boldsymbol{v}_i = \boldsymbol{v}_p + \frac{dt^3}{6}\ddot{\boldsymbol{a}}_i + \frac{dt^4}{24}\ddot{\boldsymbol{k}}_i \tag{11}$$

7) At this point particle P_i is fully updated. We set $t_i = t_i + \Delta t_i$ and then choose a new time step Δt_i for P_i's next update according to the standard Aarseth formula [12] given by (12).

$$\Delta t_i = \sqrt{\eta \frac{|\boldsymbol{a}_p||\ddot{\boldsymbol{a}}_i| + |\dot{\boldsymbol{k}}_p{}^2|}{|\boldsymbol{k}_p||\ddot{\boldsymbol{k}}_i| + |\ddot{\boldsymbol{a}}_i{}^2|}} \quad (12)$$

8) If the global simulation clock hasn't reached the simulation end time, go back to step 3 and repeat.

E. Parallelization

Many important astrophysical problems, for example the simulation of globular clusters, involve large numbers of particles (on the order of 10^6) and require a highly accurate integration method. The direct force computation scheme, owing to its $O(N^2)$ scaling characteristics, is not normally employed for such extremely large systems. With the use of special purpose hardware such as the GRAPE-6, simulating systems on the order of 10^5 particles has become possible [11], [8]. However as with most custom hardware configurations, accessibility remains limited and thus there has been considerable effort put towards parallelizing N-body kernels for execution on general purpose parallel computers [17], [7].

There are two common strategies employed when partitioning N-body systems amongst multiple processors. In the distributed data method, each processor stores particle data for only a subset of the overall system. To compute the forces acting on particles in its local subset, the processor sends a copy to the other nodes, either via broadcast or with an overlaid logical topology as in the systolic ring algorithm [7]. Partial force contributions are computed by the receiving node, after which particles are returned to the source node for the correction phase of the integrator. In the case of the systolic ring method, partial forces are aggregated as the data cycles from neighbour to neighbour.

By contrast, in the replicated data strategy each processor has a complete copy of all particles in the system and is responsible for updating a subset of these particles. This has the advantage that each node can independently compute the forces on its local group. The downside is the higher memory consumption and the costly global synchronization at the end of each update step. For large enough values of N, the performance of the systolic ring and replicated data algorithms is comparable [7]. We thus test only against the replicated data strategy in this paper.

Unfortunately, many of the methods traditionally used to accelerate direct N-body codes appear to hinder the effective parallelization of the algorithms. In the case of individual time steps, it's possible for too few particles to be active on a given update step. In a distributed context, this could result in some processors having little or no particles to update, forcing them to wait idly until the next global synchronization.

Most parallel implementations of the Hermite scheme use some form of hierarchical or block time step [17], [7]. In this approach, time steps are constrained to certain blocks of allowable step sizes, resulting in larger groups of particles being advanced concurrently during each step.

The block time step approach is a compromise between the accuracy provided by individual time steps and the simple parallel implementation afforded by constant time steps. For situations where even a small reduction in accuracy is unacceptable, we seek a method that can retain the original benefits of the individual time step strategy while also being naturally extensible to parallel implementation.

IV. Parallel discrete event N-body simulation

A. Discrete event formulation

Although normally seen as an iterative time-stepping algorithm, numerical N-body simulations exhibit enough traits of typical discrete event problems so as to render them amenable to treatment as such. In this section we detail discrete event formulations of the N-body problem for both constant-sized and variable per-particle time steps. In the descriptions that follow, we focus exclusively on systems where all processors have a complete copy of the system's particles and are responsible for advancing a subset of the overall particles. As the processors advance their state asynchronously local copies of remote particle data may become out of date; it is the responsibility of the distributed control mechanism to ensure that causality violations are avoided.

The following two sections introduce parallel discrete event N-body algorithms, firstly for the constant time step (CTS) case and thereafter extended to handle individual time steps (ITS).

B. Discrete event CTS

The obvious and natural place to start is to treat each iteration of the integration loop as a distinct STEP event, with each such event scheduling its successor. Hence an update event at time t_1 will schedule the next update event at time $t_2=t_1+\Delta t_1$, which in turn will schedule an event at $t_3=t_2+\Delta t_2$, and so on. In the case of constant time steps this is simplified to $t_{n+1}=t_n+\Delta t$ since the size of the time step is invariable.

When considering a parallelized version of this strategy, we must also take into account the updates from neighbouring processors, thus local step events are only scheduled once all outstanding updates have arrived from the other processors. This ensures that no local step is taken before first having an up-to-date copy of the global state, without resorting to a costly global synchronization barrier. The rest of this section describes the algorithm in more detail.

```
N := set of all particles
L := set of local particles
P := set of all processors

# Event handler for step events in CTS mode.
procedure event_step_cts(Event e)
{
   # Get the timestamp of the event.
   t = timestamp(e)

   # Predict position and velocity of all particles to the update
   # time t.
   predict(N, t)

   # Calculate acceleration and jerk of all local particles at
   # time t.
   calculate_force(N, L, t)

   # Correct predicted quantities for all local particles.
   correct(L, t)

   # Pack updated position, velocity, acceleration and jerk into
   # a message.
   msg = create_message(L, t)

   # Send the message to all other nodes.
   broadcast_message(msg, P)
}
```

Figure 1. Step event for constant time steps.

The various phases of the Hermite algorithm are clearly evident in the pseudocode for the step event (Fig. 1). Operating on the local copy of the particle system, we first predict the positions and velocities of all particles to the update time *t*. We then compute the acceleration and time derivative of the acceleration for all local particles, using the predicted particle state. As before, this is the most computationally intensive part of the overall simulation. Again while only operating on local particles, we compute the higher order derivatives of the acceleration and jerk, and use them to apply the correction to the position and velocities that were predicted earlier on. At this point, the local particles have been updated.

The next portion of the step event is a divergence from the time-stepping algorithm. Recall that in the parallel discrete event simulation paradigm, processors communicate by sending time-stamped messages that have the dual effect of updating local copies of state data as well as providing information essential for local synchronization. The updated portion of each local particle, namely its position, velocity, acceleration and jerk, are packed into a message structure. The message is stamped with the current local virtual time, which corresponds to the new time at which the particle information is valid.

The reception of an incoming message containing updated particle data triggers a second type of event: the remote update (RUP) event. Each processor sends its updated state information to all other nodes, which implies that each should expect to receive P-1 such RUP messages (where P is the number of processors). Only when all outstanding messages have been processed can we proceed to the next step event, and correspondingly we refrain from scheduling the next local step event until all particles have been updated.

Fig. 2 shows the pseudocode for the RUP event handler. The function apply_rup(), which performs the actual processing of

```
N := set of all particles
L := set of local particles
P := set of all processors

# Event handler for step events in CTS mode.
procedure event_rup_cts(Event e)
{
   # Get the timestamp of the RUP event.
   t = timestamp(e)

   # Update counter that tracks the number of outstanding updates.
   remaining_updates = remaining_updates - 1

   # Apply remote update to synchronize local data (see Fig. 3).
   apply_rup(e, R)

   # Test if there are still outstanding remote updates for
   # this step.
   if (remaining_updates == 0)
   {
      # Reset the update count (expect one update from each
      # other node)
      remaining_updates = |P| - 1

      # Schedule the next event
      dt = time_of_next_step(L, t)
      schedule_step(dt)
   }
}
```

Figure 2. Remote update event for constant time steps.

the contents of the remote message, is straightforward (Fig. 3). For each remote particle in the message, we update the position, velocity, acceleration and jerk fields in our local copy. Once this is done, our local copy of the sending processor's particles has been synchronized.

We have shown so far that the discrete event method can be used to emulate the more common time-stepping approach for N-body simulation. Since all required updates are sent and eventually arrive at the destination nodes, there is no risk of deadlock. Since no new STEP events are scheduled until all remote updates have been processed, particles are guaranteed to be advanced in the correct order and no causality violations can occur.

Unlike the time-stepping approach, no costly global synchronization is required at the end of each step. This is beneficial because the performance of barrier synchronization methods can vary widely due to hardware configuration and software implementation. Nevertheless, the computational workload must be evenly balanced amongst the processors in

```
N := set of all particles
U := local update list
R := set of remote particles contained in message

# Apply remote update to synchronize local data.
procedure apply_rup(Event e, Set R)
{
   # Get the timestamp of the RUP event.
   t = timestamp(e)

   # Apply the remote update
   for r in R:
   {
      # Find local copy of this remotely-owned particle
      p = find_particle(N, r->id)

      # Copy the updated component vectors
      p->position     = r->position
      p->velocity     = r->velocity
      p->acceleration = r->acceleration
      p->jerk         = r->jerk
   }
}
```

Figure 3. Applying a RUP event to update local image of remotely-owned particles.

order to avoid situations where some CPUs wait idly for the remote updates to arrive from the slower or more heavily loaded processors. For direct evaluation methods this is achieved simply by allocating an equal amount of particles to each processor, however this task is made more difficult in the context of heterogeneous clusters where the relative clock speeds of CPUs may vary, meaning that slower processors should be allocated less work than faster processors. We therefore seek to improve on this scheme by permitting processors to do useful work while waiting for incoming messages, effectively taking advantage of additional concurrency inherent in the N-body problem that's not available when constant time steps are used.

C. Discrete event ITS

This section describes a discrete event implementation of the Hermite scheme using individual per-particle time steps (ITS). We seek an algorithm that alleviates some of the difficulties in parallelizing ITS that were discussed previously in section III.E.

The primary impediment to the efficient parallelization of the time-stepping kernel when ITS is used is that some processors may remain idle while others advance their local particles. The particles on the smallest time step are selected for updating at the start of an iteration, and since all processors need to synchronize at the end of each iteration in order to exchange updated state data, processors with no particles on the smallest step have nothing to do. The problem is exacerbated as the range of allowable time steps is expanded, as we should expect more variation amongst the time steps of the system's particles.

The question then is whether the idle processors can manage to do useful work while waiting for outstanding updates from other processors. Although the particles on the smallest time step must be advanced prior to any others, we observe that some of the information necessary to advance particles on the *next smallest* time step is already available (in general, the results of all smaller time steps are missing). For example, if the smallest time step is at time t_1 and the next smallest steps are t_2, t_3 and t_4 respectively, then we can compute the acceleration of particles on step t_2 due to the particles on steps t_3 and t_4, but not due to those particles on step t_1. Upon completion of step t_1, the updated state information is disseminated amongst the nodes and can be used to calculate the remaining force contributions for the particles on step t_2. The crux of the strategy is thus to *partially advance the particles on future time steps, completing these steps later as the required missing data arrives*. We note that such an approach is not possible if processors are forced to synchronize at the end of each step prior to commencing the next step, as is the case with the time-stepping approach.

For a given particle P_i with update time $ut_i=t_i+\Delta t_i$, we say that all particles with update times less than ut_i are *unsafe*. In other words, in order to be factored into the computation of

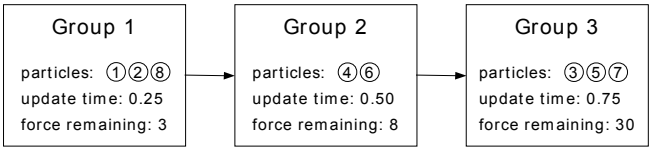


Figure 4. Update list sorted by increasing update time (ut).

P_i's acceleration at time ut_i, unsafe particles must be advanced to a time such that their next update time is equal to or greater than ut_i. Naturally, without all the required force contributions at ut_i, P_i cannot be advanced and so we say it is *blocked*. Unsafe particles can be either local or remote particles. *Safe* particles, on the other hand, are all particles with update times equal to or greater than ut_i. Note that with the ITS approach each particle has its own update time, and so the set of particles considered as safe or unsafe will in general be different from particle to particle. In the proper sequential ordering of particle updates, P_i will be updated before (or at the same time as) the particles it considers as safe. The crucial observation here is that we can calculate all force contributions on P_i at time ut_i due to safe particles, but we can only advance P_i when there are no unsafe particles relative to P_i.

To implement this strategy, each processor maintains an *update list* sorted by increasing update time. Each item in the list contains a list of local particles to update (the *update group*), the time at which these particles are to be updated, and a counter to track the number of remaining force computations necessary to completely advance the particles on the update step. Fig. 4 shows an example of a processor's local update list. The group at the head of the list represents the local particles to be updated at time t=0.25. There are three particles in this update group, and a total of three force calculations are needed to completely advance the group. Thus we can deduce that there is exactly one unsafe particle residing on some other processor that is blocking this update.

```
N := set of all particles
U := local update list

# Event handler for step events in ITS mode.
procedure event_step_its(Event e)
{
   # Get the timestamp of the event.
   t = timestamp(e)

   # Find the update group associated with this step.
   u = find_update(U, t)

   # Predict all particles that can contribute.
   safe_predict(N, t)

   # Compute partial acceleration and jerk contributions
   # from safe particles only.
   safe_force(N, u)

   # Check if all required force contributions were calculated.
   if (force_remaining(u) == 0)
   {
      # Complete this step (see Fig. 8).
      complete_updates(U)

      # Schedule future steps.
      schedule_steps(N, U)
   }
}
```

Figure 5. Step event for individual time steps.

```
# Event handler for step events in ITS mode.
procedure event_rup_its(Event e)
{
   # Get the timestamp of the RUP event.
   t = timestamp(e)

   # Apply remote update to synchronize local data.
   apply_rup(e, R)

   # Propagate RUP forward for in-progress updates (see Fig. 7).
   propagate_rup(t)

   # Scan update list for any completed updates (see Fig. 8).
   complete_updates(U)

   # Schedule future steps.
   schedule_steps(N, U)
}
```

Figure 6. Remote update event for individual time steps.

The second group in the update list (at time t=0.50) contains two particles. To complete this group eight force contributions are required, which implies that the two particles contained in this group depend on four others: the same unsafe remote particle that blocks the t=0.25 update, and the three local particles in the t=0.25 group. The final element in the list contains three local particles and has 30 outstanding force contributions, hence it depends not only on the five local particles in the t=0.25 and t=0.50 groups, but also on 5 other unspecified particles. Naturally those groups furthest in the future will have the most dependencies, since more particles will have to be updated prior to the group being advanced.

The parallel discrete event implementation of this scheme uses the same two events as in the CTS case, namely a STEP event for local particle updates and a remote update (RUP) event to incorporate incoming updates from other processors. Fig. 5 shows the general outline of a STEP event for the individual time step method. We first search the update list for the group corresponding to the time stamp of the STEP event. (The update group is guaranteed to exist as it is created and inserted into the update list when the STEP event is scheduled.) We then proceed to predict the position and

```
N := set of all particles
U := local update list
R := set of remote particles contained in message

# Propagate RUP forward for in-progress updates
procedure propagate_rup(time t)
{
   # Loop over the local update list.
   for u in U:
   {
      # The update time of the in-progress update
      ut = update_time(u)

      # Only contribute to partially done future updates
      if (started(u) AND t < ut)
      {
         # Predict the remote particles to ut.
         predict(R, ut)

         # Predict local particles in update set to ut.
         predict(u, ut)

         # Calculate acceleration and jerk components
         # due to remote particles.
         for r in R:
         {
            # Calculate force contribution of r on particles in u.
            calculate_force(r, u)
         }
      }
   }
}
```

Figure 7. Propagation of a remote update through the local update list.

```
N := set of all particles
U := local update list
P := set of all LPs

# Scan update list for any completed updates.
procedure complete_updates(Set U)
{
   for u in U:
   {
      # Complete u if there are no more force contributions
      # to calculate.
      if (force_remaining(u) == 0)
      {
         # Correct predicted positions and velocities.
         correct(u)

         # Compute next time steps for all particles in u.
         update_time_steps(u)

         # Merge completed update into other ongoing updates.
         merge_update(U, u)

         # Pack updated position, velocity, acceleration
         # and jerk into a message.
         msg = create_message(u, update_time(u))

         # Send the message to all other processors.
         broadcast_message(msg, P)

         # Remove u from the update list.
         remove_from_list(U, u)
      }
   }
}
```

Figure 8. Advancing local particles in update groups that have received all outstanding force contributions.

velocity of all *safe particles* only, and then calculate the force contributions of these particles on those in the update group. If after this operation all force contributions have been accounted for, then there were no unsafe particles to block this step, and we can advance the particles in the update group and schedule future steps. If however there are additional force calculations to be performed, then we must wait for the unsafe particles blocking this group to be advanced via RUP events and any subsequent unblocking of local particles upon which this group depends.

Remote update events for the ITS scheme (Fig. 6) are likewise slightly more involved than in the CTS case. After first applying the update to synchronize the local image of remotely-owned particles, we then propagate the newly updated state data through the local update list (Fig. 7). This has the effect of adding more force contributions to the in-progress local updates, since the particles updated as part of this RUP event must have been classified as unsafe for any update group with update time after the time stamp of this event.

When there are no remaining force contributions to be calculated on the particles of a local update group, we can complete this update step (Fig. 8). We first apply the correction phase of the Hermite integrator to correct the positions and velocities predicted at the start of the STEP event. Afterwards, we compute a new time step for each of the updated local particles using (12). Care must be taken if the next update time for one of these particles matches another that's already in the local update list. In this situation, there are two options. If no force contributions have been calculated for this update group (i.e. the corresponding STEP event has been scheduled but not executed), we simply append the particle to

TABLE I. DETAILED SPECIFICATIONS FOR LINUX CLUSTERS

Name	*CPU model*	*Node Configuration*	*Clock speed*	*Memory (per node)*
krylov	AMD Opteron 2376	27 dual-socket, quad core	2.3 GHz	16 GB
alef	Intel Xeon E5410	8 dual socket, quad core	2.33 GHz	16 GB
colosse	Intel Nehalem-EP	x6275 blades (dual-node, dual socket) 7680 cores total	2.8 GHz	24 GB

the update group and continue. However if the update group has already been partially advanced, then we must ensure that the same force calculations are applied to the particle before it's added to the group. In other words, the particle must catch up to this group prior to joining it. Merging particles into an update group is a two phase operation. First we compute the safe forces on the newcomers, and then we compute the forces on the update group due to the newcomers. This ensures that all the particles in the update group will have interacted with exactly the same list of particles, which in turn guarantees that no force contributions will be missed later on.

V. PERFORMANCE OF THE DISCRETE EVENT METHOD

In this section we compare the performance of the time-stepping and discrete event implementations of the individual time step N-body algorithm.

A. Experimental Setup

Data was generated on three Linux clusters, *krylov*, *alef*, and *colosse*. The krylov cluster is actually a heterogeneous system consisting of 21 quad-core nodes (2.2 GHz) and 27 eight-core nodes (2.3GHz), although we restrict our usage to the 8-core nodes in order to ensure that execution times between runs remain comparable. All computations were performed in double-precision floating-point arithmetic.

B. The Plummer model

We use the well known Plummer model for all experiments (Fig. 9). The Plummer model is a spherically symmetric, centrally concentrated potential model created to fit observational data [3]. We use the system of standard N-body units whereby the total mass of all particles, the radius of volume containing all particles, and the gravitational constant G are all set to unity [9]. We measure wall clock time for the integration of one N-body time unit, referred to as the crossing time, which is proportional to the number of particles in the system.

Initially, the total energy of the Plummer sphere is E=-1/4 [1]. As the simulation progresses, the finite precision of floating-point numerical calculations causes this value to drift; measuring this drift at periodic intervals is a common method of evaluating the accuracy of an N-body integrator. We compute the total energy every 1/8 of a crossing time. The results of the time stepping code are perfectly reproducible,

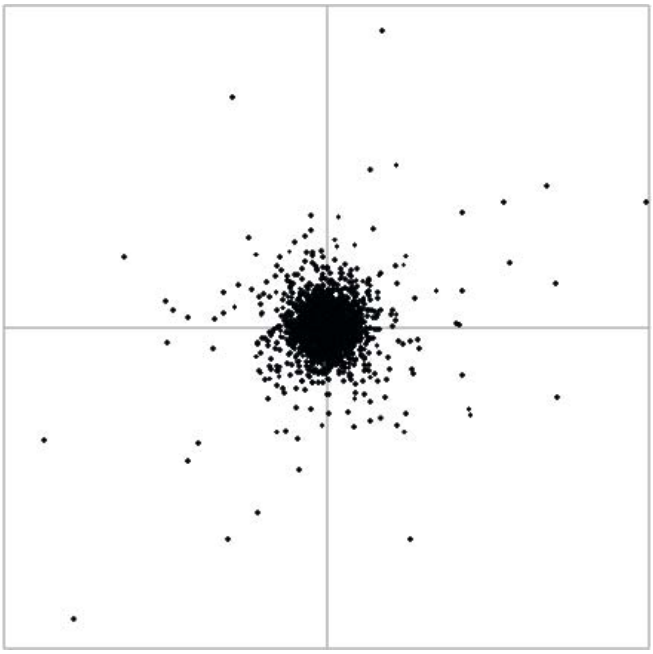


Figure 9. Plummer model (N=16384)

whereas since the discrete event method may process events in a different order from one run to the next (without, of course, introducing any causality violations), the total energy values may vary slightly. We have failed to find any significant difference in energy drift between the two methods, confirming the viability of the discrete event approach and validating the correctness of both codes.

C. Performance analysis

Fig. 10 shows a comparison between the discrete event (DE) and time-stepping (TS) algorithms. The maximum time step was set to $t_{max}=1/4$ and the minimum time step to $t_{min}=1/512$. We measured the total execution time on *alef* using 8 and 16 CPUs, for particles systems of size 2^{10} to 2^{14}. The total execution time increases dramatically with the number of particles, illustrating the problematic $O(N^2)$ nature of the direct

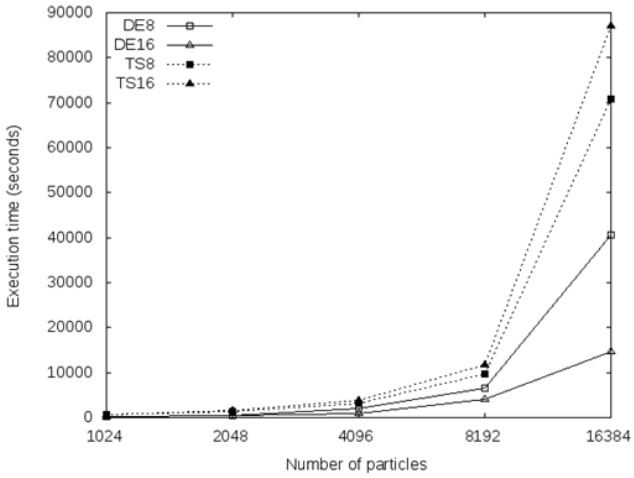


Figure 10. Scaling of the discrete event and time stepping methods on alef.

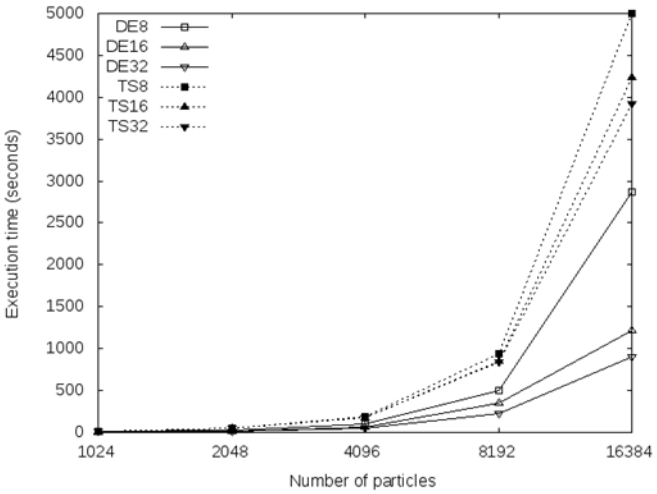


Figure 11. Scaling of the discrete event and time stepping methods on krylov.

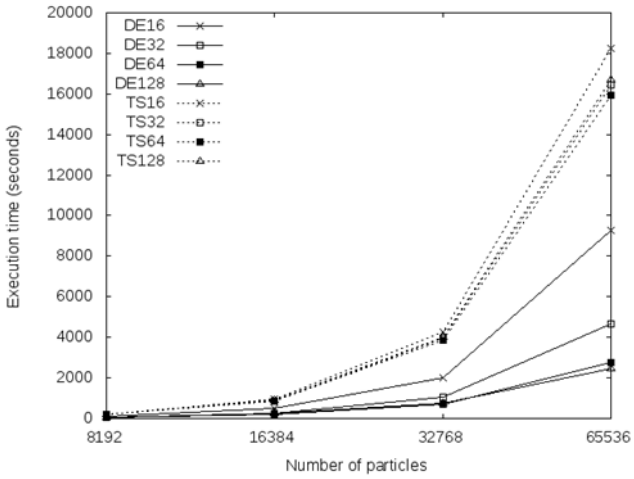


Figure 12. Scaling of the discrete event and time stepping methods on colosse.

evaluation method for N-body problems. The discrete event algorithm (shown by the solid lines in all the graphs that follow, as opposed to the dashed lines for time-stepping) scales better as the number of particles increases. It is particularly interesting to note that the performance of the time-stepping algorithm is worse for 16 processors than for 8, implying that synchronization overhead and wasted CPU cycles nullify any performance benefits seen when using the individual time step scheme on a single processor. Additionally, since the nodes of *alef* contain 8 cores, performing a barrier synchronization is clearly more expensive once more than 8 processors are involved, resulting in this surprising performance reversal when 16 processors are used. The discrete event method, on the other hand, is better able to utilize the extra processors, resulting in a 64% reduction for N=16384 when the number of processors is doubled.

Fig. 11 shows the results of a similar run on the *krylov* cluster. A larger minimum time step of t_{min}=1/128 was used to demonstrate how both algorithms fare when faced with a smaller range of particle step sizes. We also tested 8, 16 and 32 processor configurations, again restricting ourselves to the 8-core nodes. Both algorithms perform better as more processors are added, however the improvement is more evident in the discrete event case. Overall the discrete event algorithm outperforms the time-stepping strategy by a significant margin. In particular, even with only 8 processors the discrete event code manages to be almost 27% faster than the time stepping run with 32 processors. It appears that the discrete event method is better able to exploit the additional concurrency not available to methods that rely on global synchronization and lock-step execution.

The next example (Fig. 12) demonstrates how both approaches scale as the number of processors is increased. Data for this run was generated on *colosse*, the biggest of the three Linux clusters. Each data point displayed represents an average over four runs with identical parameters. The dashed lines representing the execution times of the time stepping method are all grouped together; clear evidence that adding additional processors does not yield any performance improvements in this case.

There is much more variety for the solid lines representing the results of the discrete event runs. For N=65536 particles the discrete event code outperforms the time stepping code by approximately 49%, 72%, 83% and 85% when 16, 32, 64 and 128 processors are used, respectively. The processors are kept busier by partially computing particle updates with the information on hand, and do not waste nearly as many cycles idling. The cost of performing a global barrier synchronization increases with the number of processors, which is another factor behind the poor performance of the time stepping code.

Although adding more processors does reduce the total execution time for the discrete event algorithm, we note in Fig. 12 that there is little difference between runs using 64 and 128 processors. In order to determine if larger particle systems are

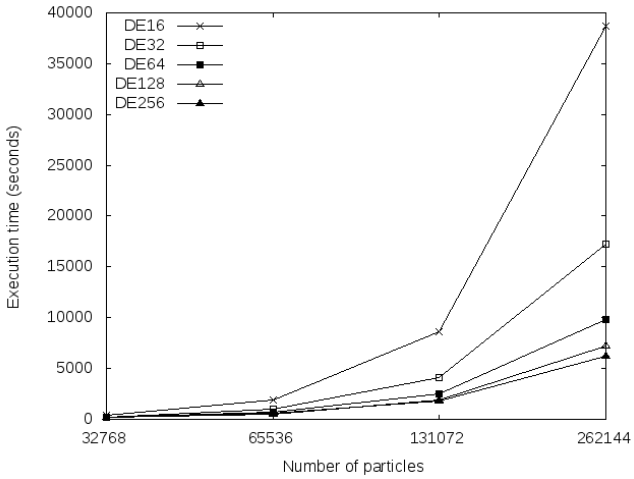


Figure 13. Scaling of the discrete event algorithm for larger particle systems.

required for performance to continue to scale well with the number of processors, we increased the maximum number of particles to 2^{18}. To reduce the overall simulation time, the minimum time step was increased to $t_{min}=1/16$. Fig. 13 shows the results for 16 to 256 processors (as before, execution time is the average of four runs). For the largest particle systems tested, we observe successive speed-ups of approximately 44%, 43%, 27% and 14% each time the number of processors is doubled, from the initial 16 to a total 256 CPUs.

VI. Future work

We believe the novel application of parallel discrete event simulation techniques presented herein provides an overture to additional applications within the domain of numerical N-body methods. In particular, we plan on extending our method to handle a variation of the Hermite integrator that employs particle-centric neighbourhoods to divide the force acting on a particle into *near* and *far* components, as described by Makino and Aarseth [12]. We plan on further extending our scope to encompass the spatial partitioning trees used in the Barnes-Hut scheme [2].

VII. Conclusion

We have described a discrete event implementation of an N-body solver using individual per-particle time steps. Whereas the use of individual time steps tends to accelerate sequential N-body simulations, in practice it is often an impediment to the efficient parallel implementation of N-body kernels. We have shown that the discrete event implementation using an application-specific control protocol outperforms the time stepping implementation and scales better with the number of processors. In various other application domains such as molecular dynamics and rigid body simulation, event-driven methods have been employed successfully to accelerate sequential simulations. Despite this potential, event-driven schemes are often seen as more difficult to parallelize than traditional time stepping approaches. We believe the work presented herein will lead to simple, efficient parallel discrete event simulations in these and other domains.

References

[1] S. J. Aarseth, "Gravitational N-body simulations," Cambridge University Press, Cambridge, UK, 2003.

[2] J. Barnes, P. Hut, "A hierarchical O(N log N) force-calculation algorithm," Nature, 1986, ISSN: 0028-0836, vol. 324, pp. 446-449.

[3] J. Binney, S. Tremaine, "Galactic Dynamics," Princeton University Press, Princeton, NJ, 1987.

[4] K. M. Chandy, J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," Communications of the ACM, 1981, Vol. 24, No. 11, pp. 198-206.

[5] R. Fujimoto, "Parallel and Distributed Simulation Systems," John Wiley & Sons Inc., New York, NY, 1999.

[6] R. Fujimoto, "Parallel Discrete Event Simulation," Communications of the ACM, 1990, Vol. 33, No. 10, pp. 30-53.

[7] A. Gualandris, S. P. Zwart, A. Tirado-Amos, "Performance analysis of direct N-body algorithms for astrophysical simulations on distributed systems," Parallel Computing, 2007, ISSN: 01678191, vol. 33, pp. 159-173.

[8] S. Harfst, A. Gualandris, D. Merritt, R. Spurzem, S. P. Zwart, P. Berczik, "Performance analysis of direct N-body algorithms on special-purpose supercomputers," New Astronomy, 2007, vol. 12, Issue 5, pp. 357-377.

[9] D. C. Heggie, R. D. Mathieu, "Standardized Units and Time Scales," In P. Hut and S. McMillan, eds, The Use of Supercomputers in Stellar Dynamics, 1986, Springer-Verlag, Berlin, pp. 233.

[10] D. Jefferson, "Virtual Time," ACM Transactions on Programming Languages and Systems, 1985, Vol. 7, No. 3, pp 404-425.

[11] J. Makino, T. Fukushige, M. Koga, K. Namura, "GRAPE-6: The massively-parallel special-purpose computer for astrophysical particle simulation," Publications of the Astronomical Society of Japan, 2003, ISSN: 0004-6264, vol. 55, part 6, pp. 1163-1188.

[12] J. Makino, S. J. Aarseth, "On a Hermite Integrator with Ahmad-Cohen Scheme for Gravitational Many-Body Problems," Publications of the Astronomical Society of Japan, 1992, 44, pp. 141-151.

[13] J. Makino, "Optimal Order and Time-Step Criterion for Aarseth-Type N-Body Integrators," The Astrophysical Journal, 1991, 369, pp. 200-212.

[14] J. Misra, "Distributed Discrete-Event Simulation," ACM Computing Surveys, 1986, Vol. 18, No. 1, pp. 39-65.

[15] R. Spurzem, "Direct N-body simulations," Journal of Computational and Applied Mathematics, 1999, Vol. 109, Issues 1-2, pp. 407-432.

[16] Y. Tang, K. S. Perumalla, R. Fujimoto, H. Karimabadi, J. Driscoll and Y. Omelchenko, "Optimistic Simulations of Physical Systems Using Reverse Computation," SIMULATION, 2006, vol. 82, issue 1, pp. 61-73.

[17] S. P. Zwart, S. McMillan, D. Groen, A. Gualandris, M. Sipior, W. Vermin, "A parallel gravitational N-body kernel," New Astronomy, 2008, Vol. 13 Issue 5, pp. 285-295.

Modeling Billion-Node Torus Networks Using Massively Parallel Discrete-Event Simulation

Ning Liu
Computer Science Department
Rensselaer Polytechnic Institute
Troy, NY 12180
Email: liun2@cs.rpi.edu

Christopher D. Carothers
Computer Science Department
Rensselaer Polytechnic Institute
Troy, NY 12180
Email: chrisc@cs.rpi.edu

***Abstract*—Exascale supercomputers will have millions or even hundreds of millions of processing cores and the potential for nearly billion-way parallelism. Exascale compute and data storage architectures will be critically dependent on the interconnection network. The most popular interconnection network for current and future supercomputer systems is the torus (e.g., k-ary, n-cube). This paper focuses on the modeling and simulation of ultra-large-scale torus networks using Rensselaer's Optimistic Simulator System (ROSS). We compare real communication delays between our model and the actual torus network from the Blue Gene/L using 2,048 processors. Our performance experiments demonstrate the ability to simulate million to billion-node torus networks.[1] The torus network model for a 16-million-node configuration shows a high degree of strong scaling when going from 1,024 cores to 32,768 cores on Blue Gene/L with a peak event-rate of nearly 5 billion events per second. Finally, we demonstrate the performance of our torus network model configured with 1-billion-nodes using up to 16,384 Blue Gene/L processors.**

I. Introduction

We are moving closer to the era of exascale (i.e., 10^{18} FLOPs) supercomputing. As machines grow larger and larger, the internal communication between millions of cores is one of the key factors that will determine future computation and storage performance for massively parallel applications. Torus networks have been widely employed as the underlying network topology for many supercomputing systems, such as the Blue Gene [1] and Cray XT5 [2] families. A torus is chosen because it yields low latency for nearest neighbor processors, scalable bisection bandwidth, and provides an easy physical wiring plan for upgrading a system with additional nodes without having to update the entire core torus network [1].

As part of the exascale co-design process, there is significant interest in understanding how parallel systems software like MPI/MPIO and the associated supercomputing applications will scale on future architectures. To this end, researchers have turned to massively parallel discrete-event simulation. For example, Perumalla's $\mu\pi$ system will allow MPI programs to be transparently executed on top of the MPI modeling layer and simulate the MPI messages. Here, each MPI task is realized as a thread in the underlying μsilk simulator. Thus, $\mu\pi$ captures the true direct execution behavior across millions of MPI tasks that are part of a whole supercomputing application. In particular, $\mu\pi$ has executed an MPI job that contained over 27 million tasks and was executed on 216,000 Cray XT5 cores. This is the largest direct execution simulation of any parallel program we are aware of to date. This is a level of scaling that similar systems, like BigSim [3] have not achieved. However, neither of these systems performs packet-level simulations of the underlying network at scale to the best of our knowledge. Instead the focus of their research is application computation performance with the network abstracted away.

The focus of our research is to create a "good" fidelity, flexible, large-scale torus network model for the purpose of understanding the performance implications of different exascale storage architectures which unlike today's current supercomputer systems, could be more closely woven into the compute-side architecture (i.e., bring storage closer to computation in order to improve performance and fault tolerance). Consequently, we need the ability to model an exascale compute network as part of our exascale storage co-design research. One use case scenario of the torus network model is determining speed by which failure information is distributed throughout the storage system across different failure detection algorithms, such as a gossip algorithm [4]. In this case, we need a good model of network congestion but a cycle accurate network simulation is too costly to process at the network scale which we are investigating, especially for longer running I/O workloads which span many massively parallel jobs and can last 12 to 24 hours if not days [5].

The central contribution of this paper is that we present the capability to execute extremely large-scale torus networks models at a "good" level of fidelity using a Blue Gene/L supercomputer. We want determine how big of a torus network model that can be executed using relatively modest supercomputing hardware (i.e., Top 500 rank is less than 100 and dropping fast). We observe that the 70 teraFLOP Blue Gene/L resource used here will be a very small slice of the upcoming "Mira" (nearly 800,000 cores) and "Sequoia" (1.6 million cores) Blue Gene/Q systems [6], [7], [8], [9] which will be available in late 2011/early 2012. These systems will provide 10 and 20 petaFLOPs of compute power respectively and greater than 200 teraFLOPS of compute power in a single rack form factor. Thus, in the near future, parallel computations consisting of 32,000 MPI tasks will be common place.

The remainder of the paper is organized as follows: Section II presents the details of torus network developed for massively parallel execution. Section III briefly introduces our parallel simulator and simulation platform. Performance experiment setup, results and analysis are presented in Section IV. Finally, related work is discussed in Section V with closing remarks in Section VI.

II. Torus Network Model

A. Torus Structure

A torus network topology is a k-ary n-cube where k is referred to as the radix and n as the dimension [10]. This grid structure has n dimensions and each dimension has k nodes.

A 3-ary 3-cube torus network is shown in Figure 1. Each of the $N(N = k^n)$ nodes can be identified by an n-digit radix k address $(a_1, a_2, \ldots, a_n)$. The i-th digit of the address vector, a_i, represents the node position in the i-th dimension. Nodes with address

[1] In this paper, we use "node" to denote a logical node in our simulation model. In contrast, "processor" and "cores" are used to denote the actual simulation platform except for the Blue Gene/L introduction subsection.

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

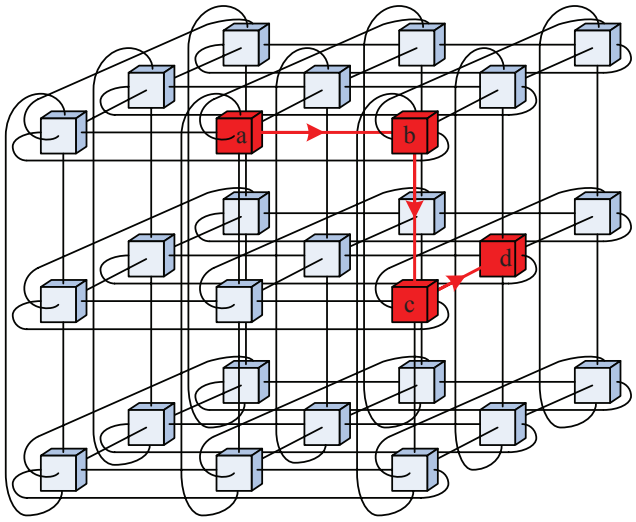


Fig. 1. 3-ary 3-cube Torus.

$(a_1, a_2, \ldots, a_n)$ and $(b_1, b_2, \ldots, b_n)$ are connected if and only if there exists i, $(1 \leq i \leq n)$, such that $a_i = (b_i \pm 1) \bmod k$ and $a_j = b_j$ for $a \leq j \leq n$; $i \neq j$.

The distance between 2 nodes in the torus is measured by number of hops. A "hop" is the process of moving data from one node to its direct neighbor. Different from a k-ary n-cube mesh, whose maximum distance between any 2 nodes is kn, the actual maximum distance in the torus is cut by half, $kn/2$. Observe that the maximum number of hops in a torus network determines the maximum latency of the communications.

Blue Gene and Cray XT supercomputer families adopt a 3-D torus. Typically, the 3-D torus network will fix the number of nodes in 2 dimensions so the system only grows in the 3rd dimension as racks of new systems are added. This leads to a linear increase in the maximum latency. One solution to this problem is to create a torus network with a larger number of dimensions as is the case with the upcoming Blue Gene/Q which will have a 5-D torus network [6], [7], [8], [9].

B. Torus Routing

By design, a torus network will provide low latencies and high bandwidth at a moderate cost to construct. We observe that a number of research efforts have concentrated on the design and optimizations of switching fabric and routing algorithms [11], [12], [13], [14] for torus networks.

As shown in Figure 1, if node a needs to send a packet to node d, it has to choose among many possible paths. One of the possible paths is labeled in the graph. Routing in torus can be carried out by either a specific routing algorithm, such as the deterministic e-cube algorithm, or a generic routing algorithm, such as the well-known up/down routing algorithm or the flexible routing algorithm [15]. In this paper, we use the deterministic static routing instead of dynamic routing or a mixture of static and dynamic which is used in Blue Gene system. Static, deterministic routing resembles e-cube routing in many ways. The well-known e-cube algorithm routes packets in decreasing dimension order to avoid deadlocks. When applied to a torus, the e-cube algorithm requires two virtual channels to remove any cyclic channel dependencies introduced by the wrap around links [15], [16]. Additionally, the use of bidirectional channels within a link between two nodes allows packets to be routed through minimal paths.

To illustrate how the e-cube algorithm works in torus, we assume that each physical channel is split into two virtual lanes (VL0 and VL1). A packet arriving at a node n_c and destined to node n_d, with coordinates $n_{c_{n-1}}, n_{c_{n-2}}, \ldots, n_{c_1}, n_{c_0}$ and $n_{d_{n-1}}, n_{d_{n-2}}, \ldots, n_{d_1}, n_{d_0}$, respectively, will be routed through the physical channel belonging to the dimension i, where i is the position of the most significant digit in which addresses n_c and n_d differ. In each dimension i, packets will be routed through VL1 if the i-th digit of the destination address is greater than the i-th digit of the current node address. Otherwise, the packet will be routed through VL0.

C. Torus traffic

Markovian models have been a popular approach understanding the interconnection network performance [10]. For the purpose of this performance study, we capture the time independent nature of the packet stream and simply let it follow the Poisson Process with a mean arrival rate of λ. In our model, each node continuously generates a Poisson stream of packets, and each generated packet randomly chooses a destination node, which follows a uniform distribution. This yields a pathological traffic pattern with little to no locality and as a consequence presents a model that is challenging to obtain good parallel performance.

According to Little's Law, if the average service time is fixed, the packet arrival rate determines the number of packets alive in the system at a steady state. For torus network model considered here, each node has an infinite queue subject to the memory limitations of the parallel computer system executing the discrete-event model such that a high packet arrival rate can saturate the system.

Based on the above discussions, we have the following assumptions for our torus model:

1) Each and every node will participate in generating, processing and relaying packets throughout the entire simulation with no failures;
2) The connection between any nodes in each direction is in good condition throughout the entire simulation with no failures;
3) Routing between nodes is static and deterministic. Note, our model is deadlock free torus given the above 2 assumptions;
4) Bandwidth is constant on each connection between the nodes for a given message size;
5) Each node continuously generates a packet stream with an exponential inter-arrival time and the destination node is selected at random.

D. Model Configuration

In Figure 2, the torus network flow chart illustrates the event driven approach on a per node basis. At the initial state, each node, modeled as a logical process (LP), will schedule a packet_generate_event shown below in Algorithm 1. When processed, the generate_packet_event will then schedule the packet_send_event immediately and another packet_generate_event with an exponential time delay. This event generation delay forms the Poisson packets stream on each node. Prior to scheduling the packet_send_event, the destination node ID is placed into packet header for the purpose of routing at each hop within the torus network. Also, the packet generation time is stored in header which enables the model to capture the end-to-end latency. We note that the current model assumes that the "application-level" message is the same size as a packet. However, in practice for real network workloads, such as the storage system, this would not be the case. Here, a single application level message could span over multiple packets. This feature of the model will be added

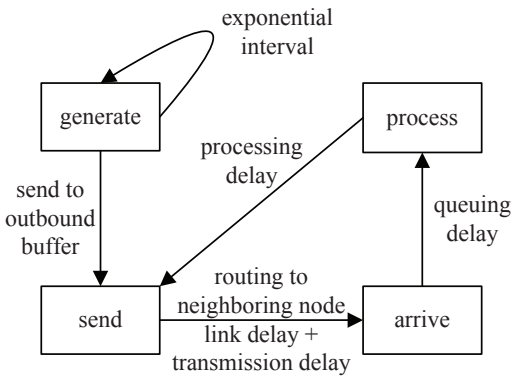


Fig. 2. Discrete-Event Torus Network Model.

at a later time when it is integrated with other proposed exascale storage architecture model components. Because of the flexibility of our model, each node can generate different packet streams and destinations in the torus model. We observe that the torus model is a sub-model that can be part of a greater application's communication model that is able to generate a traffic model in response to higher-level computation such as a file system in a proposed exascale storage architecture.

Algorithm 1 packet_generate_event.

1: generate a random time delay T_D (exponential distribution);
2: schedule next packet_generate_event after T_D on current node;
3: select random destination (uniform distribution) for current packet;
4: record packet destination in packet header;
5: record packet generating time in packet header;
6: schedule a packet_send_event on current node with delay T_D;

In the packet_send_event shown in Algorithm 2, the packet header is parsed and the next-hop node/LP is chosen. In the torus network, the next-hop node is one of the direct neighbors among the various dimensions supported in the torus based on the previously described e-cube routing algorithm. Next, the packet_arrive_event is scheduled to the target neighboring node with properly computed time delay. The time delay includes the link delay for virtual channel setup and transmission time for the packet based on the bandwidth capacity of the link. We note that the packet size on Blue Gene/L is usually a multiple of 32 bytes with a maximum size of 256 bytes [17] including the header. In our model, the packet size is initially set as 256 bytes. Packet size and bandwidth of the link determine the transmission delay.

Algorithm 2 packet_send_event.

1: compute link delay and transmission delay T_D;
2: **if** (link available)
3: schedule a packet_arrival_event on next-hop node after T_D;
4: **else**
5: acquire link next_available_time;
6: schedule a packet_arrival_event on next-hop node at next_available_time;
7: update link next_available_time;

The packet_arrive_event shown below in Algorithm 3 processes the arriving packet on the current node. Whenever a packet arrives at a node and cannot be immediately routed to the next hop node, it is placed into an incoming buffer along that dimension. In this model, each torus node has two buffers for each of its dimension. Here, one buffer is used for the $+$ direction and the other for $-$ direction for each dimension of the torus. The queuing delay largely depends on the number of packets waiting in the buffer. Currently, the waiting queue simply follows a first-come-first-serve policy. The packet_arrive_event will then schedule a packet_process_event for the next available time on the current node.

Algorithm 3 packet_arrival_event.

1: **if** (processing unit available)
2: schedule a packet_process_event on the current node with delay T_D;
3: **else**
4: acquire processing unit next_available_time;
5: schedule a packet_process_event on current node at next_available_time;
6: update processing unit next_available_time;

In order to simplify the switching fabric, we use a constant processing time for each packet. This processing time corresponds to the time used for parsing the packet header and acquiring next-hop information. The packet_process_event, shown in Algorithm 4, invokes the routing routine and determines to which neighboring node the packet will be routed. The next-hop information is packed into packet header so that packet_send_event can efficiently parse it. The hopping corresponds to an activation of packet_send_event with a processing time delay. If the destination node is the current node, the arriving packet will no longer activate any more events. Its life cycle has ended and we begin to collect the statistics such as the latency and trace. Packet processing event is illustrated in Algorithm 4.

Algorithm 4 packet_process_event.

1: **if** (packet arrived)
2: collect statistics;
3: **else**
4: analyze packet header and acquire destination node;
5: call routing routine and compute next hop node;
6: compute packet processing time T_P;
7: schedule a packet_send_event after T_P on current node;

III. ROSS, Blue Gene/L and Reverse Computation

ROSS is geared for running large scale parallel discrete-event simulation models using Jefferson's Time Warp event scheduling mechanism [18]. Here, the parallel simulator consists of a collection of logical processes or LPs, each modeling a node in the torus network. LPs communicate by exchanging timestamped event messages. In this study, this event message is modeled as packet. Like most existing parallel/distributed simulation protocols, we assume LPs may not share state variables that are modified during the simulation. The synchronization mechanism must ensure that each LP processes events in timestamp order to prevent events in the simulated future from affecting those in the past. The Time Warp [18] mechanism uses a detection-and-recovery protocol to synchronize the computation. For the recovery, we employ reverse computation which is described below.

A. Blue Gene/L

The Blue Gene philosophy holds that more powerful processors are not the answer when it comes to winning the massively parallel scaling war. Instead, the Blue Gene/L architecture balances the computing power of the processor against the data delivery speed of the network. This philosophy led designers to create smaller, lower power compute nodes comprising two 32-bit IBM PowerPCs running at only 700 MHz with a peak memory per node of 1 GB. Each Blue Gene rack is composed of 1,024 dual processor "node" cards that are divided into 32 drawers of 32 nodes per drawer. Additionally, there are specialized I/O nodes that perform all file I/O. Normally there is one I/O node for every 32 compute nodes. The Blue Gene/L system used in this performance study is a 16-rack, 32,768-processor system located at Rensselaer's Computational Center for Nanotechnology Innovations (CCNI).

B. ROSS Implementation

The ROSS API [19], [20], [21], [22] is kept very simple and lean. Developed in ANSI C, the API is based on a logical process or LP model. Services are provided to allocate and schedule messages between LPs. A random number generator library is provided based a Combined Linear Congruential Generator (CLCG). Each LP by default is given a single seed set. All memory is directly managed by the simulation engine. Fossil collection is driven by the availability of free event memory and its frequency is controlled with tuning parameters. The event-list priority queue can be configured to be either a Calendar Queue[23] or Splay Tree [24]. For this study, the Splay Tree is used in all performance experiments.

C. Reverse Computation

Reverse computation [19] is used to undo incorrect event computations. Shown in Figure 3 is the reverse handler code for all torus model event types.

As can be seen from the reverse code handler, the packet_generate_ event, packet_arrive_event, and packet_send_event routines all have straight forward state reversal operations without any control flow changes. The packet_process_event needs to know if this event reached its final destination node so the corresponding LP state variables and random numbers are correctly undone.

IV. Simulation Results

The performance study examines the impact of processor/core count on four primary metrics: (i) committed event-rate, (ii) percentage of remote events, (iii) efficiency and (iv) secondary rollbacks. We note that efficiency is defined as one minus the ratio of rolled back events divided by the number of total committed events. This is a more restrictive form of event efficiency used in [20] and a better predictor of event-rate and overall speedup. These metrics when taken together provide a more comprehensive performance picture of how the torus network model is effecting overall parallel simulator performance. Specifically, we are using the committed event-rate and not packet rate so we can observed how different torus configurations effect simulator performance since the packet rate can be effected by changes in the torus configuration.

Using these performance metrics, we focused the strong scaling study on a 16-million-node torus and then the 1-billion-node torus performance experiments. Our scaling study is based on the Blue Gene/L architecture using up to 32,768 cores, while the validation study used an SMP system with 12 cores (Intel Xeon x5650, 2.67 GHz) to save supercomputer usage, where each core supports two threads of execution (24 threads total). We begin with our validation study described next.

```
void
rc_event_handler(nodes_state * s, tw_bf * bf,
                 nodes_message * msg, tw_lp * lp)
{
  int index =
    floor(N_COLLECT_POINTS*(tw_now(lp)/g_tw_ts_end));
  switch(msg->type)
    {
    case GENERATE:
      N_generated_storage[index]--;
      s->packet_counter--;
      tw_rand_reverse_unif(lp->rng);
      tw_rand_reverse_unif(lp->rng);
      break;
    case ARRIVAL:
      s->next_available_time =
         msg->saved_available_time;
      tw_rand_reverse_unif(lp->rng);
      msg->my_N_hop--;
      s->N_wait_to_be_processed--;
      msg->queueing_times -= s->N_wait_to_be_processed;
      s->node_queue_length[msg->source_direction]
                          [msg->source_dim]--;
      msg->my_N_queue -=
         s->node_queue_length[msg->source_direction]
                             [msg->source_dim];
      break;
    case SEND:
      s->source_dim = msg->saved_source_dim;
      s->direction = msg->saved_direction;
      s->next_link_available_time[s->direction]
                                 [s->source_dim]=
        msg->saved_link_available_time[s->direction]
                                       [s->source_dim];
      tw_rand_reverse_unif(lp->rng);
      break;
    case PROCESS:
     if ( bf->c3 == 0 )
        {
          N_finished--;
          N_finished_storage[index]--;
          total_time -= tw_now(lp) -
            msg->travel_start_time;
          total_hops -= msg->my_N_hop;
          total_queue_length -=
            msg->my_N_queue;
          queueing_times_sum -=
            msg->queueing_times;
        }
      s->node_queue_length
        [msg->source_direction]
        [msg->source_dim]++;
      s->N_wait_to_be_processed++;
      break;
    }
}
```

Fig. 3. Reverse Code Event Handler for Torus Network Model.

A. Validation study

There are two parts to the validation study. First, the torus network model agrees with Little's Law under a variety of torus configurations and packets arrival rates. Second, we compared MPI_Send()/MPI_Recv() latency times of the actual Blue Gene/L network using 2K processors (1,024 node torus, 1x32x32) and

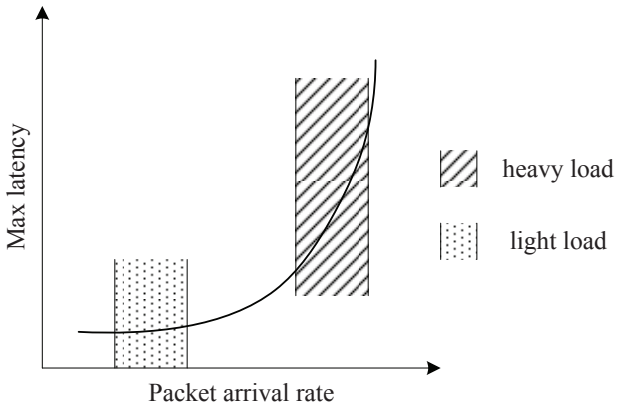


Fig. 4. Latency Prediction Curve. Predicted max latency as a function of packet arrival rate.

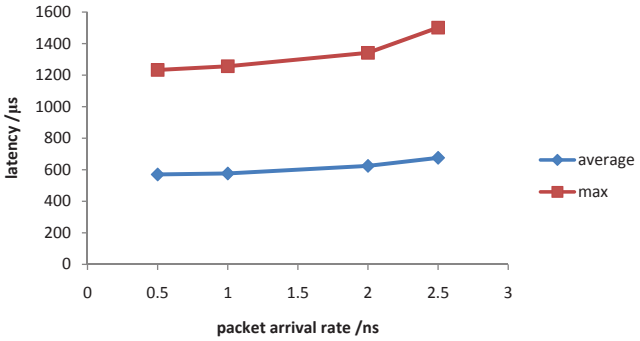


Fig. 5. 512-ary 2-cube Torus Latency.

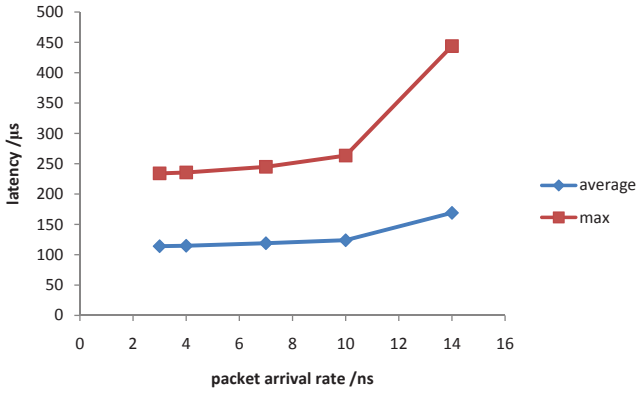


Fig. 6. 64-ary 3-cube Torus Latency.

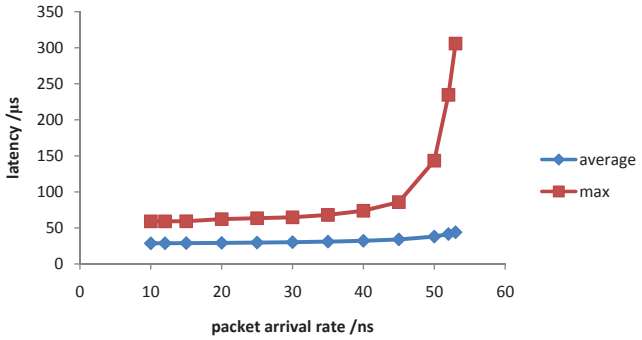


Fig. 7. 8-ary 6-cube Torus Latency.

experimental runs of our torus network model for the same torus configuration.

Starting from 256K nodes, we have built 3 different torus networks which are 8-ary 6-cube, 64-ary 3-cube and 512-ary 2-cube for the Little Law validation. The maximum distance decreases with an increase of the number of dimensions. The maximum distances for the above 3 network topologies are 24, 96 and 512 respectively. As we probe the saturation points of different torus network, the latency curves are shown in Figures 5, 6 and 7. The latency is measured in microseconds and the packet arrival rate is measured on a per nanosecond basis. These curves have a similar shape as the predicted latency curve shown in Figure 4 indicating that the queuing behavior of the model is operating as expected.

For each of the torus configurations, we captured the average and maximum latency for all packets generated during the entire simulation. The total number of packets alive during steady state is also recorded. Under a small packet arrival rate, the system arrives at a steady state very quickly. The torus network is far from saturation under this light load as there is little packet queuing on each node. The latency does not increase while the packet arrival rate increases initially. Also, average latency is in proportion with the maximum distance in different torus configurations. As the packet arrival rate continues to increase, the total number of packets alive in the system increases as well. The queuing effect leads to a dramatic increase in the maximum latency. For each configuration, there is a saturation point for a certain packet arrival rate.

For the 16-million-node torus, we varied the configuration from 8-ary 8-cube to 16-ary 6-cube and finally 64-ary 4-cube. The Little's Law results are shown in Table I. For the 1-billion-node torus the Little's Law results are shown in Table II, we varied the network configuration from 8-ary 10-cube to 32-ary 6-cube and finally 64-ary 5-cube. The experiment quickly reaches steady state with a light load of 0.5-2 pkt/ns. Observe that for both scenarios the computed latencies via Little's Law and the simulated latencies are in close agreement. Little's Law latency is computed using the number of packets waiting in the torus network and the packet arrival rate. These network models are operating in the light load zone of the latency curve shown in Figure 4.

TABLE I
16-MILLION-NODE TORUS CHARACTERISTICS: SIMULATION VS. THEORY.

torus configuration	#packets waiting	packet arrival rate	simulated latency	computed latency
8^8	168,201	4	42,022	40,700
	210,240	5	42,043	42,048
16^6	2,531,687	40	63,260	63,292
	3,166,931	50	63,306	63,339
64^4	5,095,405	30	169,908	169,847
	6,805,740	40	169,571	170,144

TABLE II
1-BILLION-NODE TORUS CHARACTERISTICS: SIMULATION VS. THEORY.

torus configuration	#packets waiting	packet arrival rate	simulated latency	computed latency
8^{10}	105,505	2	52,688	52,753
32^6	64,109	0.5	126,710	128,218
64^5	107,135	0.5	209,686	2,142,709

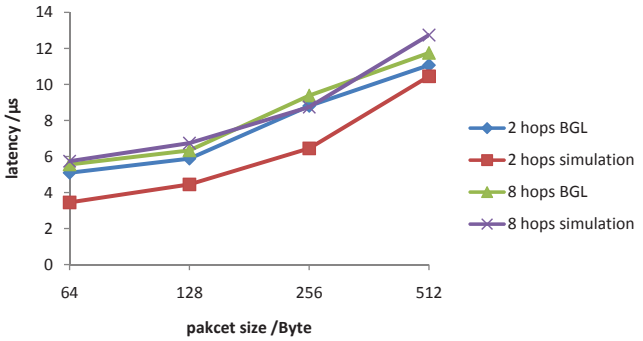


Fig. 8. Comparison of Torus Network Latency: Blue Gene/L torus VS. simulation.

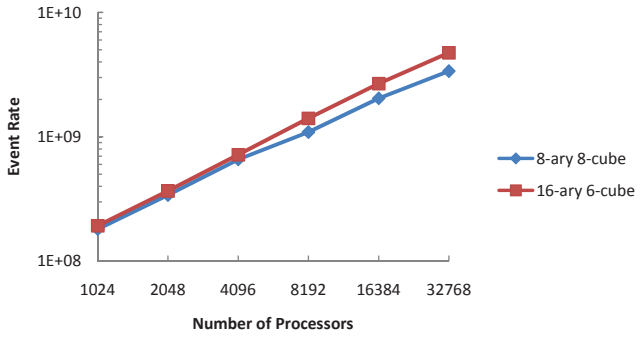


Fig. 9. Event-Rate Scalability: The event-rate is shown as a function of the number of Blue Gene/L processors.

In order to further validate the torus model, we conducted MPI_Send()/MPI_Recv() tests on Blue Gene/L and compared them to the simulation using our model as shown in Figure 8. Specifically, we measure communications times using an MPI "ping-pong" program and compare these latency measures to those predicted by our torus network model since MPI_Send()/MPI_Recv() communication operations traverse over the Blue Gene's 3-D torus network. For this comparison, we configured the torus message size to be 250 bytes which ensures use of the eager protocol being employed on the Blue Gene's real torus network and deterministically routes messages. We also varied the distances between the two communicating nodes in the torus from 2 to 8 hops. We observe the torus network model delays are matched to that of the Blue Gene/L machine with the 2 hop case and yielding more error than the 8 hop case. We believe the difference lies in some routing optimizations being made by the Blue Gene network enabling it to exceed the network performance of what our model predicts. More investigation is required to more fully understand this phenomenon. We note that because the model does perform congestion queuing correctly, we believe it will be sufficient at predicting storage architecture performance. For example, the time to commit data to RAM disk, SSD or disk ranges from hundreds of microseconds to milliseconds vs the torus network model error of 2 microseconds for a two hop torus network message. So, the current network model latency error is negligible for the exascale storage modeling purpose.

B. Million-Node Torus Scaling Study

For the scaling study, we configured two torus models with 16 million nodes. The first torus is configured as a 8-ary 8-cube, and the second is a 16-ary 6-cube. The packet arrival rate on each node is 10 pkt/ms. To reach a steady state, we set the simulation time to 100 ms. In fact, we observed the system reaching a steady state in less than 10 ms. On the Blue Gene/L, the strong scaling experiments start from using 1,024 cores and run up to 32K cores. The committed event-rate is shown in Figure 9. We observe a near linear speedup using 1,024 cores as the base case with a peak event-rate of 4.78 G/sec on 32K cores. The torus model creates a difficult scenario for parallel event scheduling because each packet randomly selects its destination which leads to a high number of remote/off-processor scheduled events. As shown in Figure 10, 11 and 12, with an increase of the number of processors, the event efficiency decreases, and the remote event-rates and secondary rollback event rates increase. At 32K cores, the event efficiency actually drops below 90% for both configurations. It is observed that the efficiency drops significantly as we decrease the system packet arrival rate. This phenomenon is

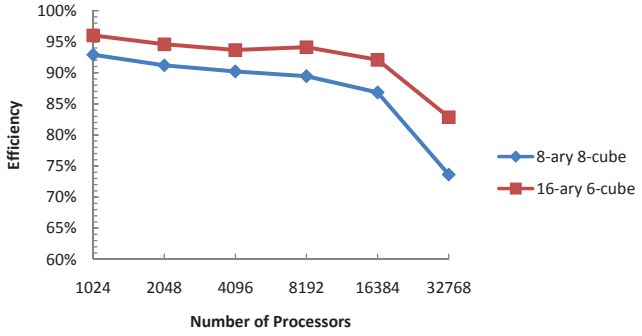


Fig. 10. Event Efficiency: Model efficiency is shown as a function of the number of Blue Gene/L processors.

attributed to a reduction in the available work per unit of simulation time which increases the likelihood of incurring a rollback.

The best event efficiency is achieved at near the saturation point of the torus network model. We speculate on the maximum latency curve shown in Figure 4 based on our experimental findings. The shaded area corresponds to the near saturation of the system or heavy load while the dotted area represents a light load. For the two torus configurations, the 8-ary 8-cube has larger bisection bandwidth and therefore smaller latency. The 16-ary 6-cube torus tends to be more saturated under the same packet arrival rate and yields a higher event efficiency and committed event-rate.

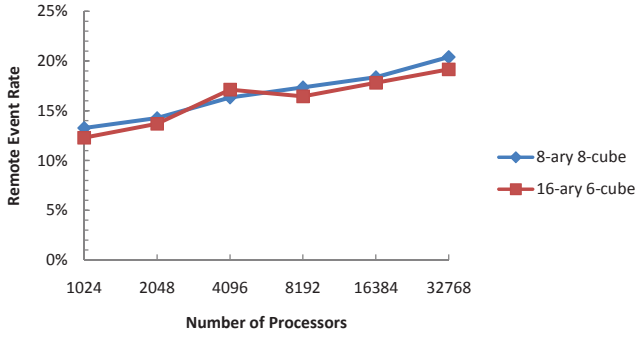


Fig. 11. Remote Events: The percentage of remote/off-processor events is shown as a function of the number of Blue Gene/L processors.

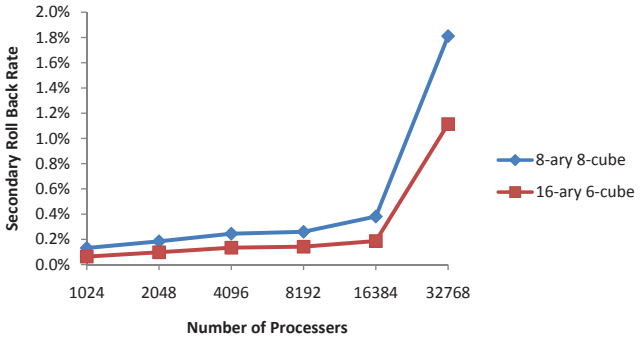


Fig. 12. Secondary Rollback Rate: The percentage of secondary rollbacks is shown as a function of the number of Blue Gene/L processors.

TABLE III
STRONG SCALING PERFORMANCE OF 1-BILLION-NODE MODEL AT CONFIGURATION 32^6 WITH PACKET ARRIVAL RATE OF $200pkt/ns$.

number of processors	4,096	8,192	16,384
number of packets (G)	40	40	40
efficiency	97.05%	96.00%	81.90%
event-rate (M/sec)	639	1,066	1,681
remote event percentage	11.72%	12.41%	13.79%
secondary rollback rate	0.0286%	0.0347%	0.220%
number of event (G)	5,644	5,644	5,644

C. Billion-Node Torus Network Scaling Study

As our model grows to 1 billion nodes, the simulation becomes more memory demanding. Consequently, the required base case (i.e., minimum number of processors that are able to execute this model) begins at 4,096 processors on Blue Gene/L with a total memory of 2TB and goes to 8,192 and 16,384 processors. The results are shown in Table III and IV.

Here, the total number of generated packets is O(10^{11}) , and the total number of events scheduled is O(10^{13}). This extreme-scale torus model can sustain a continuous packet stream of 10^{11} pkt/sec. Compared to our 16-million-node torus model experiments, the efficiency appears lower at 16,384 processors for the 200 pkt/ns scenario. This phenomenon is attributed to the 1-billion-node torus model being under-loaded with packets relative to the 16-million-node torus model. Thus, in the absence of queuing effects, events are scheduled more closely together in simulated time in the 1-billion-node torus model, leading to a higher rollback probability. The overall loss in event-rate performance is attributed to the larger event population leading to increases in queuing overheads as well as larger cache-memory overheads because of the increased amount of RAM required to execute the 1-billion-node torus model.

TABLE IV
STRONG SCALING PERFORMANCE OF 1-BILLION-NODE MODEL AT CONFIGURATION 32^6 WITH PACKET ARRIVAL RATE OF $400pkt/ns$.

number of processors	4,096	8,192	16,384
number of packets (G)	80	80	80
efficiency	97.33%	96.81%	96.42%
event-rate (M/sec)	638	1,241	1,966
remote event percentage	11.72%	12.41%	13.79%
secondary rollback rate	0.0268%	0.0312%	0.0245%
number of event (G)	11,442	11,442	11,442

V. RELATED WORK

There is a large body of previous research focusing on the modeling and simulation of torus interconnection networks. Min and Ould-Khaoua [25] proposed a torus network model based on circuit switching. Here, the traffic generated on each node follows an independent stochastic process with a given arrival rate. We follow a similar traffic pattern but our model is based on packet switching. In circuit switching, a handshaking protocol between the source and destination node which is used to initiate the communication operation. Instead of establishing a virtual channel first between source and destination, we consider incoming and outgoing buffers on each node. They also investigate long-range dependence and traffic correlation and are able to validate their model on a small scale torus network.

J. C. Sancho etc. [15] focus on the comparison of routing algorithms over torus networks. Their InfiniBand network model is composed of a set of switches and hosts which are all interconnected by a single link. The model provides the e-cube, up/down and flexible torus routing algorithms. They demonstrate the flexible routing algorithm is the most effective on small scale 3D and 2D torus networks. In our model, we have shown the effectiveness of static deterministic routing algorithm over up to 10 dimensions and extremely large scale torus network. In J. C. Sanchos work, the traffic model is simplified to switches. The packet latency is determined by changes in the switch traffic.

In the context of network simulation for supercomputer systems, Heidelberger's use of the YAWNS protocol [1] to model the Blue Gene/L torus network on a per cycle basis appears to be one of the most accurate models created to date. This work focused on the design and implementation of the 3D Blue Gene/L torus network. Details of the physical system are given which include the variable packet size, packet header size, different virtual channels and usage, per link bandwidth and different routing strategies. In contrast to the work here, we assume a fixed packet size, and for simplicity adopt static routing instead of a mixed static/dynamic routing in our simulation. In their simulation, hot region traffic and alltoall traffic are studied on a 4K node and a 32K node torus networks. We note here that our goals and intended use of this model are to understand the trade-offs associated with different exascale storage architectures and filesystems capabilities. To this end, we believe a packet-level network model such as the one discussed here will be sufficient.

Guirguis et. al. [26] examine dynamic routing over content addressable networks and demonstrate that routing can be improved while recording fewer states of the neighbors. A detailed analysis of the congestion behavior on Blue Gene/P interconnection networks is provided by P. Balaji et al. [27]. Their experiments execute on over 128K cores providing insight into the network performance characteristics of MPI.

Rahman et. al. [28], [29] examine the performance of the Hierarchical Torus Network (HTN) under the traffic pattern generated by the matrix transpose computation. They design a deadlock-free routing algorithm for the HTN using virtual channels and evaluate the performance of the HTN incontrast to the performance of conventional mesh and other networks. There research results demonstrate the high throughput and low latency capabilities of HTN over other networks.

Safaei et. al. [30] present a new mathematical model to capture the mean message latency in the torus interconnect network with circuit switching in the presence of a hot-spot. Simulation experiments demonstrate close agreement between the observed network behavior and those obtained by the analytical model.

Additional recent large-scale parallel discrete-event performance studies include Bauer et. al. [20] where an extensive study of the

PHOLD model and an electromagnetic wave propagation model is conducted. On Blue Gene/P, they report a peak event-rate of 12.3 G/sec using 65,536 cores. They have observed near linear scaling on Blue Gene/L from 1,024 cores to 32K cores. Finally, Yaun etc. [22], [31] demonstrate that optimistic protocols are able to efficiently simulate large-scale TCP scenarios for realistic, network topologies using an inexpensive Hyper-Threaded computing system.

VI. Conclusion

This paper focuses on the simulation of large scale torus networks. Based on a Rensselaer's Optimistic Simulation System (ROSS), we are able to achieve a near linear speedup for our torus model. On Blue Gene/L, the peak event-rate on 32K cores is 4.78 G/sec. We further demonstrate the ability to model a million-node and billion-node torus network on Blue Gene/L. The experiment results are validated by Little's Law for different torus configurations under varying packet arrival rate. Finally, we conducted comparison tests between actual Blue Gene torus network and our model using MPI_Send()/MPI_Recv(). The latencies captured are comparable. We believe the difference is due to subtle differences and complexities in the Blue Gene's packet routing that are not accounted for in our model. In the future, we plan to work on improving the accuracy of the model, such as introducing different torus routing algorithms and modeling the details of virtual channels.

References

[1] N. R. Adiga, M. A. Blumrich, D. Chen, P. Coteus, A. Gara, M. E. Giampapa, P. Heidelberger, S. Singh, B. D. Steinmacher-Burow, T. Takken, M. Tsao, and P. Vranas, "Blue gene/l torus interconnection network," *IBM J. RES. & DEV.*, vol. 49, no. 2/3, pp. 265–276, 2005.

[2] A. S. Bland, R. A. Kendall, D. B. Kothe, J. H. Rogers, and G. M. Shipman, "Jaguar: The world's most powerful computer," in *Compute the Future, CUG 2009 Proceedings*, Atlanta, USA, May 2009.

[3] G. Zheng, G. Gupta, E. Bohm, I. Dooley, and L. V. Kale, "Simulating Large Scale Parallel Applications using Statistical Models for Sequential Execution Blocks," in *Proceedings of the 16th International Conference on Parallel and Distributed Systems (ICPADS 2010)*, no. 10-15, Shanghai, China, December 2010.

[4] A. Das, I. Gupta, and A. Motivala, "Swim: Scalable weaklyconsistent infection-style process group membership protocol," in *In Proc. Intnl. Conf. Dependable Systems and Networks (DNS)*, 2002, pp. 303–312.

[5] A. W. Leung, S. Pasupathy, G. Goodson, and E. L. Miller, "Measurement and analysis of large-scale network file system workloads," in *USENIX 2008 Annual Technical Conference on Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2008, pp. 213–226. [Online]. Available: http://portal.acm.org/citation.cfm?id=1404014.1404030

[6] DOE, ""Sequoia" draft statement of work," https://asc.llnl.gov/sequoia/rfp/, 2008.

[7] HPCWire, "Lawrence livermore prepares for 20 petaflop blue gene/q," http://www.hpcwire.com/features/Lawrence-Livermore-Prepares-for-20-Petaflop-Blue-GeneQ-38948594.html, 2009.

[8] T. Register, "IBM uncloaks 20 petaflops bluegene/q super," http://www.theregister.co.uk/2010/11/22/ibm_blue_gene_q_super/, 2010.

[9] T. Blog, "About 16 and 17 core processors," http://www.top500.org/blog/2010/12/06/about_16_and_17_core_processors, 2010.

[10] A. Agarwal, "Limits on interconnection network performance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 398–412, 1991.

[11] G. Mora, J. Flich, J. Duato, P. Lopez, and E. Baydal, "Towards and efficient switch architecture for high-radix switches," in *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS'06)*, San Jose, California, USA, Dec. 2006, pp. 11–20.

[12] H. Abu-Libdeh, P. Costa, and A. Rowstron, "Symbiotic routing in future data centers," in *ACM Conference on Special Interest Group on Data Communication (SIGCOMM'10)*, New Delhi, India, Aug. 2010, pp. 51–62.

[13] J. Shalf, S. Kamil, L. Oliker, and D. Skinner, "Analyzing ultra-scale application communication requirements for a reconfigurable hybrid interconnect," in *Proceedings of the 2005 ACM/IEEE Super Computing (SC'05)*, Seattle, Washington, USA, Nov. 2005.

[14] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta, "Microarchitecture of a high-radix router," in *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA'05)*, Madison, Wisconsin, USA, Jun. 2005.

[15] J. C. Sancho, A. Robles, P. Lopez, J. Flich, and J. Duato, "Routing in infiniband(tm) torus network topologies," in *Proc. IEEE International Conference on Parallel Processing (ICPP'03)*, Valencia, Spain, 2003.

[16] G. M. Thorson and S. L. Scott, "Adaptive routing mechanism for torus interconnection network," U.S. Patent 421 566, Dec. 23, 1997.

[17] M. Blumrich, D. Chen, P. Coteus, A. Gara, M. Giampapa, P. Heidelberger, S. Singh, B. Steinmacher-Burow, T. Takken, and P. Vrannas, "Design and analysis of the bluegene/l torus interconnection network," IBM Thomas J. Watson Research Center, NY, Tech. Rep. RC23025(W0312-022), Dec. 2003.

[18] D. R. Jefferson, "Virtual time," *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 3, pp. 404–425, 1985.

[19] C. D. Carothers, K. S. Perumalla, and R. M. Fujimoto, "Efficient optimistic parallel simulations using reverse computation," *ACM Trans. Model. Comput. Simul.*, vol. 9, no. 3, pp. 224–253, 1999.

[20] D. W. Bauer, C. D. Carothers, and A. Holder, "Scalable time warp on blue gene supercomputers," in *Proc. ACM/IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS'09)*, Lake Placid, USA, 2009.

[21] A. Holder and C. D. Carothers, "Analysis of time warp on a 32,768 processor ibm blue gene/l supercomputer," in *2008 Proceedings European Modeling and Simulation Symposium (EMSS)*, 2008.

[22] G. Yaun, C. D. Carothers, and S. Kalyanaraman, "Large-scale tcp models using optimistic parallel simulation," in *Proceedings of the seventeenth workshop on Parallel and distributed simulation (PADS'03)*, Sandiego, USA, Jun. 2003.

[23] R. Brown, "Calendar queues: a fast 0(1) priority queue implementation for the simulation event set problem," *Commun. ACM*, vol. 31, pp. 1220–1227, October 1988. [Online]. Available: http://doi.acm.org/10.1145/63039.63045

[24] D. D. Sleator and R. E. Tarjan, "Self-adjusting binary search trees," *J. ACM*, vol. 32, pp. 652–686, July 1985. [Online]. Available: http://doi.acm.org/10.1145/3828.3835

[25] G. Min and M. Ould-Khaoua, "Prediction of communication delay in torus networks under multiple time-scale correlated traffic," *Performance Evaluation*, vol. 60, pp. 255–273, 2005.

[26] M. Guirguis, A. Bestavros, and I. Matta, "Routing tradeoffs inside a d-dimensional torus with applicability to can."

[27] P. Balaji, H. Naik, and N. Desai, "Understanding network saturation behavior on large-scale blue gene/p systems," unpublished.

[28] M. M. H. Rahman and S. Horiguchi, "High performance hierarchical torus network under matrix transpose traffic patterns," in *Proceedings of the 7th International Symposium on Parallel Architectures, Algorithms and Netowrks (ISPAN'04)*, Hong Kong, China, May 2004.

[29] M. M. H. Rahman and M. G. S. Horiguchi, "Inter-processor communication performance of a hierarchical torus network under bit-flip traffic patterns," in *th International Conference on Electrical and Computer Engineering (ICECE'06)*, Dhaka, Bangladesh, Dec. 2006.

[30] F. Safaei, A. Khonsari, M. Fathy, and M. Ould-Khaoua, "Analysis of circuit switching for the torus interconnect networks with hot-spot traffic," in *Proceedings of the 2006 International Conference on Processing Workshops (ICPPW'06)*, Columbus, Ohio, Aug. 2006.

[31] G. R. Yaun, D. W. Bauer, H. L. Bhutada, C. D. Carothers, M. Yuksel, and S. Kalyanaraman, "Largescale network simulation techniques: Examples of tcp and ospf models," *SIGCOMM Computer Comunications Review Special Issue on Tools and Technologies for Research and Eduction*, vol. 33, no. 5, 2004.

Acknowledgment

This research was supported by the DOE Exascale Co-design Program, Contract #DE-SC004875 and NSF CNS NeTS Program, Contract #0435259. Rensselaer's Computational Center for Nanotechnology Innovations (CCNI) provided the Blue Gene/L computing resources.

Empirical Study on Entity Interaction Graph of Large-scale Parallel Simulations

Bonan Hou, Yiping Yao, Shaoliang Peng
School of Computer, National University of Defense Technology, 410073, Changsha, P.R. CHINA.
pipinan@gmail.com, ypyao@nudt.edu.cn, pengshaoliang@nudt.edu.cn

***Abstract*—The entity interaction graph is an important metaphor for understanding the simulation execution of complex systems on parallel computing environment. Current performance tuning techniques often explore interrelated factors affecting performance, but ignore systematic analysis on the structure and behavior of entity interactions. This paper reports an empirical study on the entity interaction graphs of three systems chosen from different domains: Internet models, molecular dynamics, and social dynamics, respectively. The results of complex networks analysis on the entity interaction graphs demonstrate that the heterogeneous distribution of connections and highly clustering are universal in these complex systems. Generally, these properties are not obvious at the system modeling stage. Moreover, mutual information theory is used to measure the "principle of persistence" as the predictability of partitioning on multiple processors. This study facilitates better understanding and quantifying of the interaction complexity and provides implications on performance tuning for parallel simulation of large-scale complex systems.**

***Index Terms*—entity interaction graph; parallel simulation; empirical study; complex systems; performance analysis**

I. Introduction

Parallel discrete event simulation is a promising approach to study large-scale complex systems through scaling the model on high performance computing platforms [1] [2]. By decomposing the model into multiple entities that could be distributed onto multiple processors, parallel simulation can exploit the parallelism in the model, hence expecting speedup or scalability. How do entities interact with each other during the simulation? What has really happened with the execution trajectory? And how to choose a suitable load balancing strategy? Answering these questions requires better understanding on the behavior and patterns of system execution.

Many large-scale complex systems, including Internet [3], protein complexes [4] and social networks [5], manifest temporal and spatial uncertainty of entity interactions. First, most complex systems contain thousands and hundreds of entities-each entity is designed with its own characteristics and behavior rules. Second, the entities with their possible interaction partners could emerge structure complexity, which plays an important role in influencing the interaction patterns. For instance, the routing protocol in the Internet will allocate an optimized shortest-path for a request, but any two transmissions may not always go through the same path [6]. Another example is that protein-protein interaction can occur in a combinatorial number of ways, resulting in myriad protein complexes [7]. This uncertainty often results in the difficulty for applying current performance tuning techniques.

The performance tuning techniques for parallel simulation largely depends on the execution behavior of specific application. For example, the efficiency of load partition will be improved provided *a priori* knowledge of load and interaction profiles. However, it is often very hard to capture a vision of interaction patterns at the modeling stage for the intrinsic complexity and uncertainty of large-scale complex systems. Even we could have all possible interactions pre-defined, the real entity interaction patterns are still out of prediction. Current performance tuning techniques often explore interrelated factors affecting performance, but ignoring systematic analysis on the structure and behavior of entity interactions.

In this paper, we conduct an empirical study on the entity interaction of large-scale parallel simulations in order to better understand and quantify the interaction complexity. Three entity interaction graphs are obtained from runtime trace of realistic parallel applications in different domains, rather than analytical or synthesized benchmarks. Then these graphs are examined with focus on their statistical characteristics. Preliminary results demonstrate that the heterogeneous distribution of interaction partners and the existence of community structure are universal to complex parallel simulations. Moreover, the mutual information theory is used to measure the predictability of partitioning. It is hoped that our work will provide implications on performance tuning for large-scale parallel simulations and foster further research.

A. Related Work

Performance analysis and prediction is an important issue for parallel discrete event simulations determining whether the benefits of multiple processors can be gained. A considerable amount of attention has been devoted to explore a large number of interrelated factors affecting performance. These factors can be broadly categorized into two types: model-level and simulator-level. The model-level factors include event granularity, lookahead and event population [8], while the simulator-level include conservative and optimistic synchronization protocols, GVT update frequency, event list, and load balancing [9] [10] [11].

The entities partition of load balancing is one of the most studied issues as it determines the computational load and communication distribution on multiple processors. One basic assumption made by load balancing approaches is the "princi-

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

ple of persistence" [12], which assumes that the computational load and communication patterns between entities tend to persist over time so that the recent past can be used to predict near future. Thus it is important to gain better understanding of the whole interaction patterns of the system, and this is the motivation for exploring performance tuning approaches. For example, spatial scatting is suitable for domains characterized by geographic "hot-spots" [13], topology aware load balancing for molecular dynamics [12] and community-aggregation based partitioning for large-scale social networks [14]. For large-scale complex systems, however, we still have little knowledge about their interaction patterns. This may be resulted from 1) it is often impossible to plan the interaction of the whole system emerging from interact rules of parts, 2) the temporal and spatial uncertainty makes the interaction unpredictable. Fortunately, the development of complex networks science [15] enables us to quantify the empirical interaction patterns with statistical physical mechanics.

Our current work is motivated to conduct empirical analysis on realistic complex system simulation from different domains. Although conducting empirical study on parallel simulation performance is not a novel idea, it has been widely used to explore the merits or parameter space on performance [8] [9] [10]. But most of current work are either built on the synthetic PHOLD benchmark or lack systematic view of the complex interaction, which resulting in weak applicability and predictability. The statistical mechanisms borrowed from complex networks science are expected to uncover some common characteristics in the entity interaction relationships. This will facilitate our understanding of the parallel simulation of large scale complex systems and provide guidance for performance tuning.

B. Our Approach and Contributions

Here, we conduct empirical study by executing parallel simulations with trace mode on a Linux cluster. For applicability, three realistic complex applications are chosen from different domains, including a computer network simulation on ROSS.Net, a molecular dynamics simulation on NAMD, and a gossip dynamics simulation on SUPE.Net. Although they are on different parallel simulation platforms, their runtime trace can record the necessary information about who at which time scheduled an interaction for some targets. The interaction relationships can be abstracted as an entity interaction graph, where a node represents an entity while a link indicates the interaction between two entities. To explore the structure characteristics of these graphs, we apply complex networks analysis on the degree distribution of entities' interaction partners. In addition, we employ community detection algorithm on these interaction graphs, and use modularity to measure the quality of community structures. Further, we introduce the mutual information theory to measure the predictability of entity partitions. It could be a good indicator of how well the predictability of the partition algorithm has.

This article makes following contributions. First, our empirical study is conducted on realistic parallel applications from different domains, rather than synthetic benchmarks. This is important for us to better understand the execution characteristics, which are fundamentals for parallel simulation study of complex systems. Some common characteristics, such as heterogeneous degree distribution and community structure, are ubiquitously found in these entity interaction graphs. This provides a unified view to understand and quantify the interaction patterns with different complex systems. Second, the community structures are analyzed and well measured, which provides guidance on entity clustering and distribution for load balancing design. Third, the introduction of mutual information theory provides a good way to measure how the degree of "principle of persistence" is, or to what extent can the past predict the future. To the best of our knowledge, we are the first to conduct empirical study on entity interaction graphs of parallel simulation with complex networks analysis view.

The rest of the paper is organized as follows. The datasets are described in Section II. The statistical characteristics of entity interaction graph are described in Section III. The community structure characteristic is presented in Section IV. The predictability measurement of entity interactions is described in section V. The discussion is summarized in Section VI, followed by Section VII outlining conclusion and future work.

II. Datasets and Method

This section describes the datasets of three representative applications, including Internet model, molecular dynamics and social gossip.

A. Internet Model on ROSS.Net

One of our datasets is obtained from an Internet simulation on ROSS.Net, which is an optimistic parallel simulation framework for large-scale Internet models [16]. ROSS.Net can be used to validate protocol design or test the protocol stability and dynamics. The source code is publicly available at $http://sourceforge.net/projects/pdes/develop$. ROSS.Net reads network topology and traffic scenario, and can scale the simulation on multiple processors. Because ROSS.Net is built on ROSS [17], an optimistic parallel discrete event simulation engine, the basic entity is logical process (LP) and the interaction is implemented as event scheduling. The network nodes and protocol layers are automatically mapped onto LPs which in turn mapped onto processors. Every LP in the whole simulation has an ID as unique identifier.

We execute the 3M-ncs scenario with trace mode. The entities in the simulation include routers and hosts, which further composed as subnets in different areas. The information of event scheduling between LPs (rather than the simulated network nodes or protocol layers) are recorded into a log file at runtime. There are totally 45,361,568 records of event scheduling captured during the simulation.

B. Molecular Dynamics Simulation on NAMD

We use a molecular dynamics simulation ApoLipoprotein-A1 (apoa1) on NAMD as another dataset. NAMD [18] is

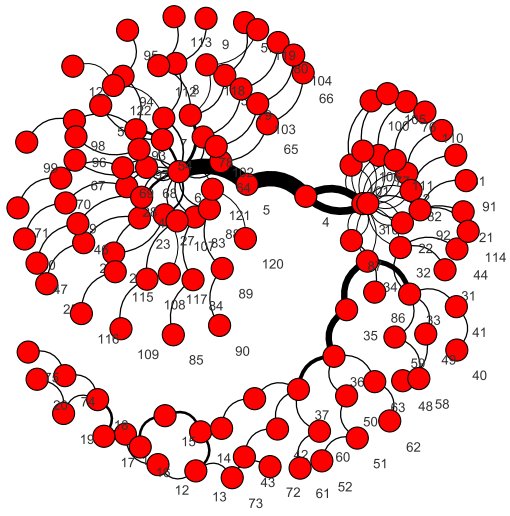


Fig. 1: The entity interaction graph of 3M-ncs simulation (in part). Each node represents a simulation LP, and the link as the interaction between two entities with the width proportional to the frequency of interaction.

a highly scalable molecular dynamics simulation using the Charm++ runtime framework [19]. The simulation contains two kinds of chares (the concept of concurrent object in Charm++): "patches" and "compute". The "patches" are cells of the 3D simulation space and the "compute" is responsible for the force calculation between every pair of patches. These entities are inherited from chare, which is message-driven and can migrate across processors. Because NAMD has employed a hybrid of spatial and force decomposition, we use the source code mol3d provided by Abhinav Bhatelé [12] as an alternative. The entity interaction information is captured by executing the apoa1 simulation with a trick option "+balancer CentralLB", which prints out the objects and communication data periodically.

C. Gossip Model on SUPE.Net

The gossip is a basic form for modeling many kinds of social dynamics, thus it is widely used in epidemic spreading, rumor/information diffusion, and opinion formation on an inter-connected population. We execute the gossip model on an actor network with SUPE.Net [20], which is an efficient parallel simulation environment designed for large-scale networked social dynamics. The basic entity in SUPE.Net is called simulation object, and the interaction between objects is implemented as event scheduling. In the model, once an individual receives a message, he/she may disseminate it to neighbors. The message will cascadingly propagate in the population. It is necessary to mention that although the dynamics flow on social network and the network structure will influence the dynamics, the real interactions between simulation objects are different from the social network itself. Because the nodes are able to choose which neighbor they would like to interact with, and some links would become hot-spot which are out of expectation.

TABLE I: Basic characteristics of interaction graphs.

Interaction Graphs	Vertices	Edges	$< k >$	Max. k	density
3M-ncs	251,703	176,845	2.00	3,759	8E-06
apoa1	7,672	16,492	4.30	90	5.60E-04
gossip	224,440	765,092	6.82	947	3.04E-05

III. Characteristics of Interaction Graph

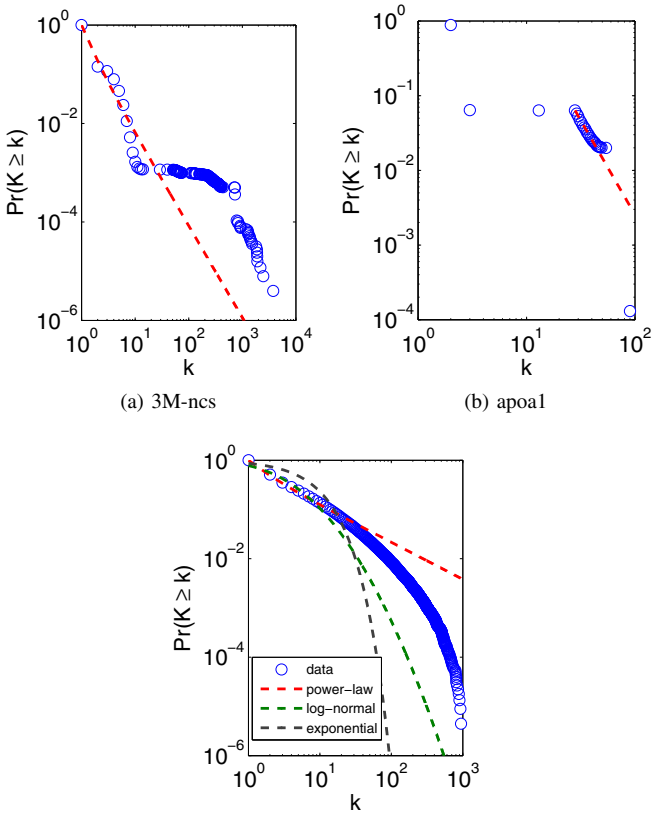
The entity interaction graphs of 3M-ncs, apoa1, and gossip simulations are constructed from the above trace datasets. Because the ROSS.Net, NAMD, and SUPE.Net adopt different concepts in the modeling framework, we use the word entity referring to the LP in ROSS.Net, the chare in NAMD, and the simulation object in SUPE.Net, and use interaction as event scheduling or message exchanging for convenience. The entity interaction graph is constructed as follows. First, we create an empty weighted graph $G(V, E)$ where V and E are sets of vertices and edges. The information about with whom who interact at which time are captured from the trace data. Each entity is represented by a vertex in the graph, and a link connects two vertices if an interaction occurs. The weight of edge indicates the strength of the interaction, which could be the sum of communication volume or the frequency of the interactions if the previous data are not available. As more nodes or links are added and the weight of links changed along with time, the whole execution of the simulation can be abstracted as an evolved weighted graph.

Figure 1 is a snapshot of the entity interaction graph of 3M-ncs simulation. Each node represents a simulation LP, and the link as the interaction between two entities with the width proportional to the frequency of interactions.

The interaction graphs we obtained are sparse, but still too complicated for human perception. The structure is irregular, and the strength of interaction is not uniformly distributed. The basic characteristics of these entity interaction graphs are listed in Table I. The degree of vertices differs greatly even they represent homogeneous entity models. The average degree $< k >$ in these graphs is relatively small, but the maximum degree is very large.

The degree k distribution of these interaction graphs is illustrated by Figure 2. The fraction of vertices with degree k, $P(k)$, versus k is plotted on a log-log scale. The lines fit the data as long-tailed distributions as contrasts. The distribution displays obvious heterogeneous and long-tailed characteristic, which implies that the majority of entities have several partners, while entity with large partners is also possible. The probability of a entity with a large number of interaction partners is rather significant, because the interaction hot-spots are often dominated by highly connected entities.

The underlying reasons for the feature of heterogeneous distribution can be explained as follows. First, it is determined by the model's intrinsic structure. For many complex systems, the most entities have several interaction partners



(a) 3M-ncs (b) apoa1

(c) gossip

Fig. 2: The complementary cumulative distribution functions $P(k)$ on degree distribution of entities interaction graph by a log-log plot. The heavy-tailed property is universal but ambiguous to characterize. The solid lines fit the data just for comparison. (a) 3M-ncs simulation on ROSS.Net, fitted by power-law [21]($x_{min} = 1, \alpha = 2.86$); (b) apoa1 simulation on NAMD, fitted by power-law ($x_{min} = 28, \alpha = 3.5$); (c) gossip model on SUPE-Net, fitted by power-law ($x_{min} = 1, \alpha = 1.75$), log-normal ($\mu = 0.87, \sigma = 1.14$), exponential ($\lambda = 6.82$).

while some may have large connectivity. In the Internet model, the numerous client hosts have fixed communication channels but the routers are more widely accessible. The situation is similar in molecular dynamics simulation. Each "compute" is responsible for the force calculation between every pair of patches, while a "patch" may have many "computes". Second, the stochastic characteristic in complex systems will result in entities with high-degree connectivity emergence. Although each entity acts with predefined behavior and optimized rules, such as choosing the shortest path or interacting with the most active neighbors, some hot-spot will form spontaneously which may not be expected at designing stage. For the gossip model, messages are apt to transform among friends, and it is more likely to loop in clusters with dense links.

TABLE II: The modularity values for the interaction graphs, indicating the structures are highly modular as modularity ≥ 0.4.

Interaction graph	Communities	Modularity
dolphins	4(62 vertices,159 edges)	0.514
zachary	3(34 vertices,78 edges)	0.381
3M-ncs	248	0.981
apoa1	10	0.622
gossip	839	0.665

IV. Community Structure and Characteristics

To measure the clustering characteristic of interaction patterns, we borrow the concept of community in social networks. The community refers to groups of nodes with a higher density of internal links, whereas links between communities with less dense. For many networks of complex systems, the community structure has great impact on their organization and function. Specifically for the entity interaction graph, the community clusters the entities which interact with each other in tightness.

The modularity is a well known quality function for estimating the goodness of community partition. It is calculated as the fraction of edges within communities minus expected fraction of such edges in the random graphs (null model), that is

$$Q = \frac{1}{2m} \sum_{C \in P} \sum_{i,j \in C} (A_{ij} - \frac{k_i k_j}{2m}), \quad (1)$$

where A_{ij} is the weight of the link between vertices i and j, and $m = \sum_{i,j} A_{ij}/2$ is the total weight in the graph [22]. In general, the modularity value of 0.4 or greater is considered meaningful, and the higher the value is, the much more modular the structure has.

We use the fast modularity maximization algorithm, known as Cluset-Newman-Moore algorithm [23], to find the potential community structures in the entity interaction graphs. The modularity values for these graphs are listed in Table II. The dolphin social network and karate club network of Zachary, which are often used as benchmarks for community detection, are used as contrasts. The table indicates that 3M-ncs, apoa1, and gossip interaction graphs have very high modularity values, indicating that these graphs have good community structures. This hints to the existence of subgroups of entities with much more internal edges.

It is necessary to mention that Table II can not be used for comparison. For instance, we can not say that the graph with higher modularity has better community structure than those with lower modularity. According to the definition, modularity should not be used to compare the quality of the community structure of graphs which are very different in size [24]. Modularity is just a good indicator for whether there exists community structure or not.

We observed that the community structure is a ubiquitous property of complex systems. This enables us to understand the systems with another dimension, which is similar to

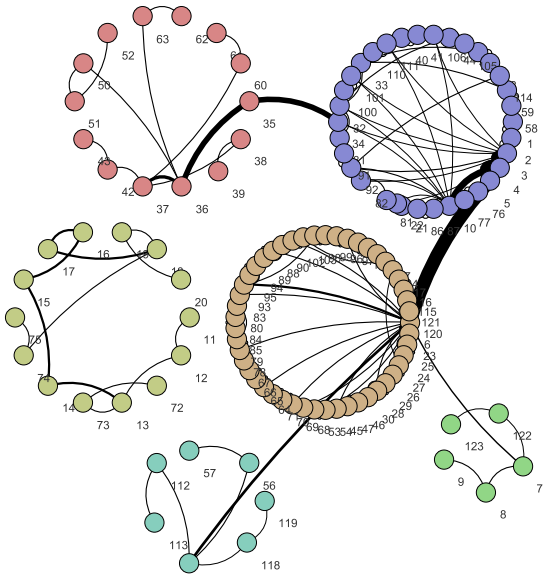


Fig. 3: The community structures of 3M-ncs interaction graph in Figure 1.

the geographic regions for spatially clustered simulations. Moreover, the community structure provides new insight into the interactions of the overall system, facilitating further researches on performance tuning techniques.

V. Predictability

Both static and dynamic load balancing strategies are built on the assumption of "principle of persistence" which uses the interaction patterns so far to predict the future. For static entity partitions, we expect the interaction pattern will not change frequently during the whole simulation execution. For dynamic load balancing, the load distribution needs to be reevaluated periodically and some entities will migrate across processors in order to optimize the benefit of balanced computing load and reduced communication volume. There is a trade-off between the expected benefit and the cost of load migration. So the decision to migrate or not largely depends on how good the persistence of the interaction patterns is. By recording the location of entities periodically during the simulation execution, we can get a series of snapshots of the whole partition. Then the simulation execution can be abstracted as time-related evolution of entity interaction graph.

Mutual information [25] between the graphs at different stages can be used to measure the capability of predicting future by using current interaction patterns. The predictability problem can be described as follows. The interaction graph at time t is denoted by $G(V, E, w, t)$, where V is the set of entities, E is the set of interaction relationship, and w is weight of edges. Given a load partitioner f, it will optimize a partition of the interaction graph as $Par(t) = f(G(V, E, w, t)) = \{P_i | \cup_i P_i, P_i \cap P_j = \phi, i \neq j\}$ for balanced computing load and minimum communication volume. For an arbitrary k partition Par, the probability that a given entity will fall in part i as $p(i) = n_i/N$, where n_i is the number of entities in part i and N is the total number of entities in the system. The mutual information $I(Par(t), Par(t'))$ between partitions at different times indicates how many entity n_{ij} of part i of partition $Par(t)$ are in part j of partition $Par(t')$, and is defined as $I(Par(t), Par(t')) = \sum_{i=1}^{q_t} \sum_{j=1}^{q_{t'}} \frac{n_{ij}}{N} log(\frac{n_{ij} N}{n_i n_j})$.

TABLE III: The normalized mutual information of the partitions at the begin and end of simulations.

Interaction graphs	2	8	32	64
3M-ncs	0.787	0.0683	0.1144	0.0641
apoa1	0.4245	0.0476	0.0106	0.0068
gossip	0.5150	0.1753	0.0000	0.0000

For convenience, the predictability of interaction graph under given partitioner f is obtained from normalized mutual information $I_N(Par(t), Par(t'))$ between two partitions, that is

$$I_N(Par(t), Par(t')) = \frac{2I(Par(t), Par(t'))}{H(Par(t)) + H(Par(t'))}, \quad (2)$$

where $H(Par(t))$ is the Shannon entropy of a partition and is calculated as $H(Par(t)) = -\sum_{i=1}^{q_t} \frac{n_i}{N} log \frac{n_i}{N}$.

We attempt to calculate the mutual information of above interaction graphs on multiple processors. As not all of these simulations support dynamic load balancing, the METIS graph partitioning algorithm [26] is used to optimize the partitions. The interaction graph without weight, which may be inferred from system design, acts as the initial partition, and the end graph as compared partition which means it is need to reevaluate the load distribution for further balancing. The results of mutual information with different partitions are illustrated by Table III.

The mutual information of partitions at the beginning and end of simulation are kept in a low level, less than 10% averagely. The results indicate that it is hard to obtain good performance using entity partition without enough *a priori* information on interactions. Moreover, the partitioning strategies should be carefully adjusted for periodically optimism as tedious cost of migration would result in. It explains the difficulty of performance tuning for complex systems from another side.

VI. Discussion

Empirical study is an important approach for understanding the simulation execution of complex systems. The results of interaction patterns among simulation entities are important for our understanding of complex systems' structure and behavior, since that may be not obvious at the designing stage. The heterogeneous distribution of connectivity and the universal community structure present implications on performance tuning techniques for large-scale parallel simulations. Current load partition and balancing approaches need to be improved, even redesigned from scratch, in order to achieve better performance on parallel computing environment by taking the advantages of these characteristics.

However, there are still some limitations. First, it may be dangerous to conclude the long-tailed distribution is an universal characteristic of complex systems' interaction. More studies on different kinds of complex systems are needed for confirmation. Second, because there is still no census on the definition of community, we expect to explore the special meaning attached to community from the aspect of entity interactions. At last, the mutual information measurement is used to describe the similarity of different partitions. It defines the theoretical bound for transformation from one partition to another. It cannot, however, measure the actual cost of migration of entities.

VII. Conclusion and Future Work

In this paper we report the empirical results on entity interaction graphs of complex systems from the structural perspective. The long-tail degree distribution is widely found, which means most of entities have several interaction partners while there exist highly interconnect entities. Moreover, the entities naturally cluster into community structure which describes their interaction relationship from mesoscopic view. The characteristics result in lower predictability of interaction patterns, which poses challenges for performance tuning. This study helps us to mine the hidden facts on interaction pattern at the modeling phase and facilitates our understanding the structure and behavior of complex system simulations.

In future work, we intend to design performance tuning techniques, such as community-based entity partitioning and mapping, under the guidance of the above empirical results. Furthermore, identifying the important entities that may dominate interactions in the simulation would be further studied for better load balancing.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (NSFC) Grant (No.60773019, 60903223) and the Ph.D. Programs Foundation of Ministry of Education of China (No. 200899980004). The authors would like to thank Abhinav Bhatelé for generously providing mol3d code, and the anonymous reviewers for their comments and suggestions.

References

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

[2] Y. Yao and Y. Zhang, "Solution for analytic simulation based on parallel processing," *Journal of System Simulation*, vol. 20(24), pp. 6617–6621, 2008.

[3] R. Winter, "Modeling the internet routing topology - in less than 24h," in *ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation (PADS '09)*, 2009, pp. 72 –79.

[4] N. J. Krogan and G. Cagney, "Global landscape of protein complexes in the yeast saccharomyces cerevisiae," *Nature*, vol. 440(7084), pp. 637–643, 2006.

[5] L. Barrett, S. Eubank, V. S. A. Kumar, and M. V. Marathe, "Understanding large scale social and infrastructure networks: A simulation based approach," *SIAM news: The Mathematics of Networks*, 2004.

[6] H. Levin, M. Schapira, and A. Zohar, "Interdomain routing and games," in *Proceedings of the 40th annual ACM symposium on Theory of computing (STOC '08)*, 2008, pp. 57–66.

[7] W. S. Hlavacek, J. R. Faeder, M. L. Blinov, A. S. Perelson, and B. Goldstein, "The complexity of complexes in signal transduction," *Biotechnology and Bioengineering*, vol. 84(7), pp. 783–794, 2003.

[8] V. Balakrishnan, P. Frey, N. B. Abu-Ghazaleh, and P. A. Wilsey, "A framework for performance analysis of parallel discrete event simulators," in *Proceedings of the 29th conference on Winter simulation (WSC '97)*, 1997, pp. 429–436.

[9] J. Liu, D. Nicol, B. Premore, and A. Poplawski, "Performance prediction of a parallel simulator," in *Proceedingsof the Thirteenth Workshop on Parallel and Distributed Simulation*, 1999, pp. 156 –164.

[10] C. Carothers and K. S. Perumalla, "On deciding between conservative and optimistic approaches on massively parallel platforms," in *Proceedings of the 2010 Winter Simulation Conference*, 2010, pp. 678–687.

[11] G. Zheng, "Achieving high performance on extremely large parallel machines: performance prediction and load balancing," Ph.D. dissertation, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 2005.

[12] A. Bhatelé, L. V. Kalé, and S. Kumar, "Dynamic topology aware load balancing algorithms for molecular dynamics applications," in *Proceedings of the 23rd international conference on Supercomputing (ICS '09)*, 2009, pp. 110–116.

[13] S. Thulasidasan, S. Kasiviswanathan, S. Eidenbenz, and P. Romero, "Explicit spatial scattering for load balancing in conservatively synchronized parallel discrete event simulations," in *2010 IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS),*, May 2010, pp. 1 –8.

[14] B. Hou and Y. Yao, "Commpar: A community-based model partitioning approach for large-scale networked social dynamics simulation," in *IEEE/ACM 14th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, 2010, pp. 7 –13.

[15] A.-L. Barabási, "Scale-Free Networks: A Decade and Beyond," *Science*, vol. 325, no. 5939, pp. 412–413, 2009.

[16] G. Yaun, C. Carothers, and S. Kalyanaraman, "Large-scale tcp models using optimistic parallel simulation," in *Proceedings. Seventeenth Workshop on Parallel and Distributed Simulation (PADS 2003)*, 2003, pp. 153 – 162.

[17] C. Carothers, D. Bauer, and S. Pearce, "Ross: a high-performance, low memory, modular time warp system," in *Proceedings of the Fourteenth Workshop on Parallel and Distributed Simulation (PADS 2000)*, 2000, pp. 53 –60.

[18] J. Phillips, G. Zheng, S. Kumar, and L. Kale, "Namd: Biomolecular simulation on thousands of processors," in *ACM/IEEE 2002 Conference Supercomputing*, 2002, p. 36.

[19] L. V. Kale and S. Krishnan, "Charm++: A portable concurrent object oriented system based on c++," in *Proceedings of the Conference on Object Oriented Programming Systems, Languages and Applications*, 1993, pp. 91–108.

[20] B. Hou, Y. Yao, B. Wang, and D. Liao, "Supe-net: an efficient parallel simulation environment for large-scale networked social dynamics." in *2010 IEEE/ACM International Conference on Green Computing and Communications & 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing*, 2010, pp. 628–635.

[21] A. Clauset, C. R. Shalizi, and M. E. J. Newman, "Power-law distributions in empirical data," *SIAM Rev.*, vol. 51, pp. 661–703, November 2009.

[22] M. E. J. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[23] A. Clauset, M. E. J. Newman, and C. Moore, "Finding community structure in very large networks," *Phys. Rev. E*, vol. 70, no. 6, p. 066111, Dec 2004.

[24] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3-5, pp. 75 – 174, 2010.

[25] T. M. Cover and J. A. Thomas, *Elements of information theory*. New York, NY, USA: Wiley-Interscience, 1991.

[26] G. Karypis and V. Kumar, "Multilevel k-way partitioning scheme for irregular graphs," *Journal of Parallel and Distributed Computing*, vol. 48, p. 96C129, 1998.

Collaborative Interest Management for Peer-to-Peer Networked Virtual Environment

Cheng Liu, Wentong Cai
Parallel and Distributed Computing Center
School of Computer Engineering
Nanyang Technological University
Singapore 639798
{LIUC0012, aswtcai}@ntu.edu.sg

***Abstract*—Interest Management (IM) aims at eliminating irrelevant state updates transmitted in Networked Virtual Environment (NVE). The traditional IM mechanisms can be classified into area-based and cell-based mechanisms. Hybrid IM mechanism was proposed recently to reduce the communication overhead by utilizing the cell-based mechanism to reduce Area-Of-Interest (AOI) updates in the area-based mechanism. Compared to the traditional mechanisms, the hybrid mechanism reduces the upload bandwidth consumption at each player site and thus allows more players to join the virtual environment with today's network capacity. However, it requires each player maintains some global information about all other players. Updating the global information undoubtedly involves communication overhead. This paper proposes a collaborative IM mechanism for peer-to-peer NVEs, which works only relying on each joining player's local knowledge rather than globally available information so as to further reduce the communication cost of IM mechanisms. The performance of our collaborative IM mechanism is evaluated by simulating multiplayer game scenarios and the results are compared with the hybrid IM mechanism as well as the well-established peer-to-peer NVE neighbor discovery mechanism VON.**

***Index Terms*—Interest Management, Networked Virtual Environment, Parallel and Distributed Algorithm**

I. Introduction

A Networked Virtual Environment (NVE) is a network application that comprises a set of interconnected nodes that cooperatively simulate the behavior of various entities in a shared virtual environment [1]. It originated with military simulators in 80's [2] and has evolved to some popular networked games such as First Person Shooter (FPS) games (e.g., Quake [3]) and Real-Time Strategy (RTS) games (e.g., Starcraft [4]). In the past few years, Massively Multiplayer Online Games (MMOG), which is also a kind of networked virtual environment, emerged and it enables hundreds or even thousands of players to cooperate and compete with each other on a large scale game world. The subscriptions of massively multiplayer online games have been increasing over the years, bringing considerable revenues and making itself the largest category in the online entertainment markets [5], [6].

The client-server architecture is widely employed in the traditional NVEs and also in most of today's commercial MMOGs. A centralized server (or multiple servers) is responsible for account management, player authentication, and most importantly maintaining all entities' status in the virtual world. Players can access the game world through connecting to the server. Every player's action that changes an entity's status in the virtual world must be firstly sent to the server, then server disseminates the entity status updates to other players. The client-server architecture can be employed easily to support NVEs, and it is good at handling security issues such as cheat prevention [7]. However, the client-server architecture lacks of efficiency and scalability because servers may easily become the computation and communication bottlenecks. Althougth many researchers use server clusters or Grid resources to make the client-server architecture support NVEs better [8], the following drawback in client-server architecture still remains. The network latency due to message forwarding through servers can lead to inconsistent views among players. Player satisfaction will be degraded if they suffer from significant delay [9]. Moreover, from the economic aspect, the high server maintenance cost in the client-server architecture may create entry barriers which prevent small game publishers from entering the market.

Peer-to-Peer (P2P) architecture emerged in the past few years and it has been applied in many networked applications, such as distributed computing [10], database systems [11], and file sharing [12], [13], [14]. As elaborated in many research works, one major advantage of the P2P architecture over client-server architecture is its scalability. The peers in the P2P architecture are allowed to send messages directly to other peers and the resources distributed in the peers can be shared among all the participating peers. Therefore, it avoids the need of a server and hence the capacity limitation in the server does not exist anymore, making the whole system scalable. Moreover, the P2P architecture usually comes with mechanisms to handle peer joining, leaving, and failures dynamically with acceptable connectivity and performance.

Many NVEs [15], [16], [17], [18] which appeared recently take the advantages of the P2P architecture. In the P2P NVEs, each player is responsible for maintaining a small portion of entities in the virtual environment and the entity status updates are disseminated in a P2P manner. The computation and communication overhead is shared among joining players so the bottleneck due to centralized entity management is removed. The network latency to transmit the status updates,

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

which has a great impact on the quality of service in NVEs, is reduced as well since the communication among players is no longer through any server.

In order to keep a consistent view among players who are participating in a NVE, frequent entity status update is required. However, due to the lack of IP multicast support in the Internet, the status updates in P2P NVEs have to be sent to other players using unicast. This may increase the communication requirement beyond a player's capacity. Two common approaches to reduce the status updates between players are dead reckoning [19], [20] and Interest Management (IM) [21], [22]. In addition to NVE, IM techniques have been widely adopted in many other research fields such as agent-based simulation [23], [24], [25] and most recently, wireless sensor network [26]. It utilizes filtering mechanism to avoid broadcasting communication among processes or network nodes. How IM is applied in P2P NVE is the focus of this paper and it is realized by defining an Area-Of-Interest (AOI) for each entity, which describes a region in the virtual environment that the entity is interested in. In this paper, we focus on avatars which are the presence of players in the virtual environment. The AOI of each avatar represents the region that the player is interested in. A player should always receive status updates (e.g., position updates) of other players' avatars which are inside the AOI of its avatar. By using IM approach, the number of status updates can be significantly reduced. The traditional IM mechanisms can be classified into cell-based mechanisms [27], [28] and area-based mechanisms [29], [30]. The recently proposed hybrid IM mechanism [31] focuses on the communication overhead in area-based IM mechanism and successfully reduces the AOI updates in the area-based mechanism by utilizing the cell-based mechanism. However, it requires each player to maintain some global information about all other players. Updating the global information undoubtedly involves communication overhead. This paper proposes a collaborative IM mechanism for P2P NVEs, which works only relying on each joining player's local knowledge rather than globally available information so as to further reduce the communication cost of IM mechanism. A multiplayer game scenario is simulated to compare the collaborative IM mechanism with the hybrid IM mechanism. Some game scenarios are designed as well to compare our collaborative IM mechanism with the well-established peer-to-peer NVE neighbor discovery mechanism VON.

The rest of this paper is organized as follows. In Section II, we will firstly review the existing IM mechanisms. Our collaborative IM mechanism will be presented in Section III. The experimental evaluation through simulated game scenarios will be shown in Section IV and related works will be also compared. At last, Section V concludes this paper and introduces our future work.

II. Problem Definition and Related Work

A. Interest Management

We consider a set of inter-connected peers that host entities moving around in a shared virtual environment (VE). Each peer periodically refreshes the status of its hosted entities based on their kinetics and user actions. The period between two successive refreshes is called a frame. Each entity has an associated AOI that moves along with the entity and the AOI is a region of any shape in the VE. Each entity's position and AOI information must be exchanged between the peers to maintain the knowledge of whether an entity is located in the AOIs of other entities. In this paper, each peer is responsible for only one entity, so the terms of peer, player and entity are used exchangeable. This is consistent with the feature of many P2P NVEs as in most of today's multiplayer online games, each player controls only one entity, i.e., its avatar. For P2P NVEs where a peer is responsible for a set of entities, some simple modifications on the proposed mechanism in this paper are needed. Moreover, for the simplicity of description, the sequel of this paper uses two-dimension VEs, but the idea of this paper can apply to VE with any number of dimensions.

The traditional IM mechanisms include cell-based mechanisms and area-based mechanisms. The cell-based IM mechanism is also known as the grid-based IM [27], [28]. The virtual environment is divided into disjoint cells generally in squares [28] or hexagons [27]. For each cell, an underlying multicast communication network is set up and each entity maps its AOI to the cells and joins the corresponding multicast groups of the cells overlapping with its AOI (referred to as *overlapping cells*). At the meantime, each entity sends its status updates to the multicast group of the cell where it currently resides (referred to as *residing cell*). The area-based IM mechanism, which is also known as the region-based IM mechanism in the distributed simulation area [29], [30], makes each entity keep updating its AOI to other entities. Upon receiving an AOI update of another entity, the receiving entity checks whether its current position in the virtual world falls in the AOI. With this exact matching result, an entity is able to send its status updates to other entities which are interested in the updates.

With the cell-based IM mechanism, an entity sends its AOI updates only when its AOI moves across the cell borders. An entity must send its status updates to a remote entity as long as the remote entity's AOI overlaps with the local entity's residing cell. Therefore, irrelevant status updates exist since an entity may send its status updates to the remote entity whose AOI actually does not include the local entity's position. Both the IM overhead and the irrelevant status updates are dependent on the cell size. Ayani et. al. developed an analytical model and derived a formula for choosing the optimal cell size in terms of both computation and communication cost [32].

The area-based mechanisms require each entity to keep updating its AOI to all other entities, so it causes too much communication and computation overhead if each entity sends its AOI update once it moves [33]. Two simple strategies are to send AOI updates periodically [33] or in a space-driven manner [21], where an entity sends a new AOI update only when its position has changed over a distance threshold since last AOI update. In order to avoid the missing status updates between successive AOI updates, both strategies require an

AOI to be artificially expanded to include the true AOIs before the next AOI update [33]. This expansion leads to irrelevant status updates since an entity may receive the status updates from other entities which are actually outside its current AOI. The updating period or the distance threshold must be carefully chosen in order to optimize the communication efficiency of the area-based mechanism.

Hybrid IM mechanism [31] reduces the communication cost in area-based IM mechanism by utilizing the cell-based IM mechanism. The entire virtual environment is partitioned into cells in squares. An entity is aware of the AOIs of other entities whose AOIs intersect with its residing cell and the area-based IM mechanism is performed when an entity moves within its residing cell. When an entity sends its AOI updates, it will use the residing cell information of other entities and decide the set of entities that should receive the AOI updates. For a target entity which updates its AOI, the communication overhead in sending AOI updates is reduced since the entities whose residing cells are outside or fully covered by both old and updated AOI of the target entity are not necessary to receive the AOI update. Shortcoming still exists in the hybrid IM mechanism that each entity's residing cell information must be available to all entities. Therefore, the communication overhead is still high since the residing cell update must be disseminated to all entities when an entity changes its residing cell.

B. Interest Management in P2P NVE

There are many existing P2P architectures to support NVEs. Some of them apply similar ideas of interest management used in agent-based simulation, that is, some specific processes are created to perform interest management in the middleware [23], [24], [25]. Instead of using specific processes in the middleware, the existing P2P architectures to support NVEs usually select some peers to perform either traditional cell-based IM mechanism or area-based IM mechanism [15], [17], [34]. The structured P2P overlays, such as Distributed Hash Table (DHT), are used to organize these peers and it is responsible for either handling the query about the responsible peer for each cell [15], [34] or supporting range query [17]. However, the structured P2P overlay network which serves as a base in these architectures involves multi-hops latency in handling IM query which could affect the consistency in NVEs.

The most relevant work to our collaborative IM mechanism is the P2P neighbor discovery mechanism called mutual notification [18], [35]. In this P2P discovery mechanism, all participating entities form an overlay networks, in which each entity establishes the direct connections to the nearby entities in the virtual environment. When an entity moves, it sends the AOI update to the connected entities which are responsible for informing the sending entity about new entities that appear in the updated AOI. VON [18] is a representative implementation of mutual notification mechanism. It adopts Voronoi diagrams to represent the neighborship of entities. For a given entity in the virtual environment, it firstly keeps connection with a set of entities called enclosing neighbors whose Voronoi regions are adjacent. Meanwhile, each entity also connects to the entities called AOI neighbors whose positions fall into its AOI. Some neighbors are determined as the boundary neighbors which have Voronoi regions partially outside the given entity's AOI. The neighbor discovery in VON works as follows: when an entity moves, its position update and AOI are sent to all currently connected neighbors. If the recipient is a boundary neighbor, it will check to see whether any of its enclosing neighbors is now visible to, or even becomes the enclosing neighbor of, the moving entity. If new neighbors are identified, notifications are sent to the moving entity for initiating connections. This P2P neighbor discovery mechanism can effectively achieve good scalability by making each entity connect to a limited number of entities. But the AOI updates which perform at each frame may result in certain communication overhead under the scenarios which will be studied carefully in Section IV-D.

III. Collaborative Interest Management

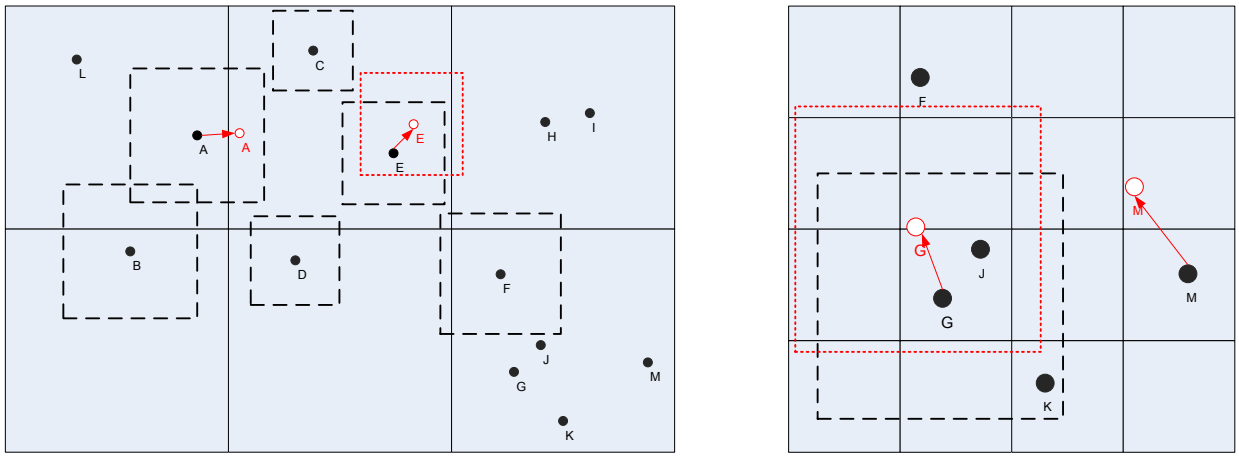
In this section we will introduce our collaborative interest management mechanism. We will introduce the 2-layer grid-based virtual environment partitioning first. Then the data structure which is maintained by each entity will be presented. After that we will present the algorithm which runs at each player side.

A. Overview

As discussed in Section II-A, the original area-based mechanisms ask each entity to send AOI updates to all other entities once the periodical or space-driven criterion is triggered. Updating the AOI information is not necessary when the remote entity is far away from the local entity's AOI. Based on this observation, the hybrid mechanism successfully reduces the communication cost of updating AOI information by making the message sender determine the necessary receivers. However, each entity needs to always update its residing cell information to all other entities when its residing cell changes.

Our collaborative IM mechanism works on a two-layered grid-based virtual environment. The hybrid IM mechanism is adopted in the second layer in order to reduce the communication cost of AOI updates, and the first layer is used to reduce the unnecessary residing cell updates in the hybrid IM mechanism. Figure 1 shows an example of our virtual environment partitioning. The entire virtual world is firstly partitioned into six cells in layer 1 (see Figure 1(a)) and each layer-1 cell is partitioned further into 16 cells in layer 2 (see Figure 1(b)). All entities' positions and some entities' AOIs are represented in the figure as solid dots and dashed squares respectively.

In our collaborative IM mechanism, each entity is always aware of other entities whose AOIs intersect with its layer-1 residing cell. When an entity's layer-2 residing cell changes due to entity's movement but its layer-1 residing cell remains the same, the layer-2 residing cell update only needs to be delivered to the entities whose AOIs intersect with sending



(a) Layer 1 partitioning

(b) Layer 2 partitioning

Fig. 1. Overview of Collaborative IM

entity's layer-1 residing cell (rather than to all participating entities in the virtual environment like in the hybrid IM mechanism). Let's take entity M in Figure 1(b) as an example. Entity M changes its residing cell in layer 2 as it moves to a new position represented by an unfilled dot, and it will only send its residing cell updates to entity F, G, J and K whose AOIs are intersecting with M's layer-1 residing cell.

When an entity is about to update its AOI information but the corresponding layer-1 overlapping cells keep unchanged, the AOI update is only performed at layer 2 using the hybrid IM mechanism. Taking entity G in Figure 1(b) as an example, its updated AOI is represented as a dotted square. Entity G sends its updated AOI information to entity F and K since both entities need the information to perform matching between entity G's new AOI and their own entity positions. The AOI updates will not be sent to entity M and J. Entity M does not require entity G's AOI information because its residing cell is outside the overlapping cells of entity G's AOI. Entity J does not need entity G's latest AOI information either because its layer-2 residing cell is covered by the overlapping cells of both entity G's previous AOI and its new AOI.

Problems remain when an entity is going to change its layer-1 residing cell or update the layer-1 overlapping cells of its AOI. When an entity changes its layer-1 residing cell, it must notify a set of remote entities about its joining of the new residing cell. The set of remote entities includes the entities whose layer-1 overlapping cells contain the new residing cell of the local entity. For example in Figure 1(a), entity A changes its layer-1 residing cell, and entity C, D, E and F should be aware of layer-1 residing cell update of entity A. Otherwise, entity A may miss necessary AOI updates from entity C, D, E and F when their AOIs move closer to entity A's position. When an entity's AOI is updated, the updated AOI may intersect with new layer-1 cells (e.g., entity E in Figure 1(a)). In this situation, the entities residing in those layer-1 cells should know the entity that is updating AOI information. Otherwise, the entity may miss the necessary layer-2 residing cell updates or even status updates of entities in its new AOI. So the challenges are how to disseminate residing cell and AOI updates in layer 1 and how to combine those with the hybrid IM mechanism implemented at layer 2. Our proposed IM mechanism aims at solving the problem of disseminating layer-1-related messages by making entities work in a collaborative manner. That is, each entity can learn the information from other entities it has already known or rely on these known entities to disseminate the messages to other entities which are currently unknown to the local entity. In other words, our collaborative IM mechanism makes it unnecessary for all entities to know each other directly. This makes the proposed mechanism significantly different from the hybrid IM mechanism.

We make assumption that the entity's moving speed in the virtual world is limited so that an entity cannot move to a certain position which is outside of its current AOI in a single frame. With this assumption, when an entity's residing cell is going to change due to entity's movement, the entity must have already been aware of some entities in its new residing cell. Therefore, the entity can send a query message to the remote entity whose residing cell is the same as the one that the local entity is about to enter. The remote entity will reply with a list of entities whose AOI is intersecting with its residing cell. Let's take entity A in Figure 1(a) as an example. Before entity A moves, it already knows entity C and E since their layer-1 residing cell is in layer-1 overlapping cells of entity A's AOI. Entity A will query one of them and get a reply including entity D and F whose AOIs also intersect with the layer-1 cell which entity A is moving into.

When an entity is going to update its AOI, the updated overlapping cells of entity's AOI may cover some layer-1 cells that are unknown to the entity previously. In this situation, the entity will select some remote entities that it already knows, and the layer-1 overlapping cells of these remote entities' AOIs contain the layer-1 cells that the local entity's AOI will intersect with. Using the message forwarding through

these selected remote entities, the AOI update can then be disseminated successfully to all entities who need this AOI update. For example in Figure 1(a), entity E is going to update its AOI shown as the dotted square in the figure. Entity E finds that one of its known entity, entity F, can help it to disseminate the AOI update message to the entities (i.e., entity H and I) in the layer-1 cell that will become one of its overlapping cells.

B. Local Data Structure

Each entity has two tables called CandidatePublisher and CandidateSubscriber. Each entry in the table CandidatePublisher keeps the information of a remote entity whose residing cell is in the local entity's layer-1 overlapping cells. The information kept for the remote entity includes the remote entity's IP address, the remote entity's layer-1 residing cell and layer-2 residing cell. Each entry in the table CandidateSubscriber keeps the information of a remote entity whose set of overlapping cells includes the local entity's layer-1 residing cell. The IP address and the corresponding layer-1 overlapping cells of the remote entities' AOIs are recorded as table entries in CandidateSubscriber.

C. Per Entity Algorithm

We will describe the collaborative IM algorithm on a per entity basis such that each entity executes the algorithm in parallel. The algorithm for each entity to execute at each frame is shown in Algorithm 1. The space-driven mechanism is adopted in our collaborative IM algorithm that an entity sends a new AOI update when its position has changed over a predefined distance threshold since the last AOI update. Similar to the area-based mechanisms discussed in Section II-A, our collaborative IM algorithm can also use the periodical mechanism. In order to guarantee the physically correct filtering, the AOI in our algorithm is actually the entity's expanded AOI which must always include the entity's true AOI before the next AOI update [33]. Each entity also runs an additional thread to handle the incoming messages. Routines to handle incoming messages are presented in Algorithms 2, 3 and 4 respectively according to message types.

When an entity starts the algorithm, its CandidatePublisher includes the remote entities who are residing in the layer-1 overlapping cells of the local entity's AOI and its CandidateSubscriber includes all remote entities whose AOIs are intersecting with the local entity's layer-1 residing cell. If the entity finds that its layer-1 residing cell changes in a frame (Step 3 in Algo. 1), it tries to find a remote entity in its CandidatePublisher, whose current position is in the layer-1 cell that the entity is moving into. Then it will send a *L1RC_UPDATE* message to that remote entity (Step 5 in Algo. 1). When the remote entity receives the message (Step 1 in Algo. 2), it will reply the sender a *L1RC_UPDATE_REPLY* message which includes its current CandidateSubscriber and the overlapping cells of its own AOI. After the entity who is about to change its layer-1 residing cell receives the message, it will update its own CandidateSubscriber accordingly (Step 5 in Algo. 2). A *RC_LEAVE* message will be sent to the remote entity who does not belong to the entity's CandidateSubscriber anymore. Some entries in the CandidateSubscriber will be updated and new entries will be also added into the table (Step 13 in Algo. 2). The entity will send a *L2RC_UPDATE* message to all the remote entities in the CandidateSubscriber to indicate that it becomes their candidate publisher (Step 18 in Algo. 2).

If the entity's layer-1 residing cell does not change, the entity will invoke the procedure which is similar to how the hybrid IM mechanism works (Step 7 to 15 in Algo. 1). The entity calculate its layer-2 residing cell and check whether the

Algorithm 1 Per Frame Algorithm

1: *make movement for the local entity*;
2: *calculate new layer* 1 *residing cell* (**newL1RC**);
3: **if newL1RC** $\neq$ *last layer* 1 *residing cell* **then**
4: *find an entity* **e** *in* CandidatePublisher *whose residing cell is* **newL1RC**;
5: *send a L1RC_UPDATE to* **e**;
6: **else**
7: *calculate new layer* 2 *residing cell* (**newL2RC**);
8: **if newL2RC** $\neq$ *last layer* 2 *residing cell* **then**
9: *send a L2RC_UPDATE to remote entities in* CandidateSubscriber;
10: **else**
11: **for** *each remote entity* **e** *in* CandidateSubscriber **do**
12: *check the current position of the local entity against* **e**$'s$ *expanded AOI information*;
13: *send a StatusUpdate to* **e** *if necessary*;
14: **end for**
15: **end if**
16: **end if**
17: **if** *position shifting from the last AOI update exceeds DistanceThreshold in either dimension* **then**
18: *calculate new overlapping cells* (**newOC**);
19: **if newOC** $\neq$ *last overlapping cells* **then**
20: **for** *each remote entity* **e** *in* CandidatePublisher **do**
21: **if** **e**$'s$ *residing cell does not belong to* **newOC** **then**
22: *send a OC_LEAVE to* **e**;
23: **end if**
24: **end for**
25: *find a set of entities* **E** *in* CandidateSubscriber *that the union of those entities' overlapping celles can cover* **newOC**;
26: *send OC_UPDATE with ttl equal to* 1 *to all entities in* **E**;
27: **else**
28: **for** *each remote entity* **e** *in* CandidatePublisher **do**
29: *send an AOIUpdate to* **e** *according to hybrid IM mechanism*;
30: **end for**
31: **end if**
32: **end if**

Algorithm 2 Handling Incoming Messages (Related to the Change of Layer-1 Residing Cell)

```
1: if type = L1RC_UPDATE then
2:    content = CandiateSubscriber
      with the information of local entity;
3:    reply L1RC_UPDATE_REPLY(content)
      to the sending entity;
4: end if
5: if type = L1RC_UPDATE_REPLY then
6:    for each remote entity e in CandidateSubscriber do
7:       if e is not in the message content then
8:          send a RC_LEAVE to e;
9:          remove the entry about e
            from CandidateSubscriber
10:      end if
11:   end for
12:   for each remote entity e in the message content
      do
13:      if e is not in CandidateSubscriber then
14:         add new entry in CandidateSubscriber;
15:      else
16:         update entry in CandidateSubscriber;
17:      end if
18:      send a L2RC_UPDATE to e;
19:   end for
20: end if
21: if type = RC_LEAVE then
22:   remove the entry in CandidatePublisher;
23: end if
```

layer-2 residing cell changes or not. When the entity finds that its layer-2 residing cell changes, it will send a *L2RC_UPDATE* message to all the remote entities in the CandidateSubscriber (Step 9 in Algo. 1). This message also includes the entity's position, so the remote entity can calculate the sending entity's layer-2 residing cell, update the CandidatePublisher, and process the position status as well (Step 11 to 17 in Algo. 4). After processing the *L2RC_UPDATE* message, the entity will reply its last sent AOI update to the sending entity if the sending entity falls into its layer-2 overlapping cells (Step 18 to 20 in Algo. 4). When an entity receives *AOIUpdate* messages, it will update its known AOI information which is maintained in its CandidateSubscriber accordingly (Step 23 in Algo. 4).

If the entity's layer-2 residing cell does not change either, the entity simply checks its new position with each remote entity's AOI in CandidateSubscriber, and sends status updates to the remote entities whose AOIs cover its new position (Step 11 to 14 in Algo. 1).

When the space-driven AOI update mechanism is triggered, an entity firstly calculates the layer-1 overlapping cells of its new AOI. Once it finds that the overlapping cells change, the entity may need to remove some remote entities from its CandidatePublisher since their layer-1 residing cell is no longer overlapping with the entity's new AOI (Step 20 to 23 in Algo. 1). The corresponding remote entities will receive the *OC_LEAVE* messages and remove the sending entity from their own CandidateSubscriber (Step 16 to 18 in Algo. 3). After that, the entity whose layer-1 overlapping cells are changing selects some remote entities from its CandidateSubscriber. The union of the overlapping cells of those selected remote entities' AOIs covers the new AOI of the entity. The entity then sends a *OC_UPDATE* message which contains the information about its new overlapping cells to the selected remote entities and the time-to-live (TTL) value of the message is set to 1 (Step 25 to 26 in Algo. 1). When a remote entity receives the message, it will forward the message to the entities in its CandidatePublisher. But, to avoid the message being forwarded again, the TTL value attached with the message is changed to 0. At last, all the remote entities who are residing in the overlapping cells of the target entity should be aware of its new layer-1 overlapping cells. They will update their own CandidateSubscriber (Step 2 to 12 in Algo. 3) accordingly. They will also send a status update back to the target entity to inform that they become the target entity's candidate publishers (Step 14 in Algo. 3). When an entity finds that the overlapping cells of its AOI keep unchanged, it will send the AOI update to some remote entities according to the hybrid IM mechanism using the layer-2 information (Step 28 to 30 in Algo. 1).

Algorithm 3 Handling Incoming Messages (Related to the Change of Layer-1 Overlapping Cells)

```
1: if type = OC_UPDATE then
2:    if ttl = 1 then
3:       change ttl value with OC_UPDATE to 0;
4:       forward the OC_UPDATE
         to entities in CandidatePublisher;
5:       update the entry in CandidateSubscriber;
6:    else
7:       if the message originator is not
         in CandidateSubscriber then
8:          add new entry in CandidateSubscriber;
9:       else
10:         update the entry in CandidateSubscriber;
11:      end if
12:   end if
13:   calculate layer 2 overlapping cells
      of the message originator;
14:   send a StatusUpdate to the message originator;
15: end if
16: if type = OC_LEAVE then
17:   remove the entry in CandidateSubscriber;
18: end if
```

IV. Experimental Evaluation

A. Simulation Setup

A P2P multiplayer game scenario is simulated to evaluate the performance of the proposed collaborative IM mechanism. The size of virtual environment is 1000 distance units by

Algorithm 4 Handling Incoming Messages (Related to Status Update, Layer-2 Residing Cell Update and AOI Update)

```
1: if type = StatusUpdate then
2:    calculate the sending entity's layer 2 residing cell;
3:    if the sending entity is not in CandidatePublisher
      then
4:       add new entry in CandidatePublisher;
5:    else
6:       update entry in CandidatePublisher;
7:    end if
8:    process the status update;
9: end if
10: if type = L2RC_UPDATE then
11:   calculate the sending entity's layer 2 residing cell;
12:   if the sending entity is not in CandidatePublisher
      then
13:      add new entry in CandidatePublisher;
14:   else
15:      update entry in CandidatePublisher;
16:   end if
17:   process the status update;
18:   if local entity's layer 2 overlapping cells include
      the sending entity's layer 2 residing cell then
19:      send an AOIUpdate to the sending entity;
20:   end if
21: end if
22: if type = AOIUpdate then
23:   update entry in CandidateSubscriber;
24: end if
```

1000 distance units. Each player controls only one entity, i.e., its avatar, and there are 600 players in the game. Each avatar is associated with a square AOI which centers at the position of player's avatar, and AOI size is defined as the length of AOI square. All players' avatars are initially placed at random in the virtual environment. A commonly used random waypoint mobility model [36], [37] is adopted to simulate avatar movement. In this model, an avatar chooses a random destination point in the VE and moves towards it. Upon arrival, the avatar chooses another random destination point and repeats the process. The moving speed of each avatar is set as 1.0 distance unit per frame. All simulation experiments were carried out on a machine with Intel dual-core CPU Q9450, 3.00 GB RAM and Windows XP OS. Each simulation run executes for a period of 5000 frames. In our experiments, we assume that players can make connections directly with each other, therefore messages can be received by players without any relay. The communication costs in terms of number of messages per frame in various mechanisms are measured and compared.

B. Success Rate of Collaborative IM

In our collaborative IM mechanism, an entity may fail to update its local data structures under certain situations. For example, when an entity changes its layer-1 residing cell, it may fail to find another entity which has been already residing in the layer-1 cell that it moves to. Similarly, an entity may also fail to find suitable entities which can help it to disseminate the updates about its new layer-1 overlapping cells. Entity's AOI size and the layer-1 cell size of the virtual environment are two important factors that affect the success rate of updating local data structures in our collaborative mechanism. Figure 2 presents how both the layer-1 cell size and entity's AOI size can affect the success rate when an entity changes its layer-1 residing cell. The layer-1 cell size is set as 200, 250, 300, 350, 400, 450 and 500 distance units. The AOI size varies from 20 distance units to 160 distance units. We can find that the success rate in updating layer-1 residing cell is not affected by the AOI size when the layer-1 cell size is fixed. The success rate of updating layer-1 residing cell in the experiments is 100 percent except when the layer-1 cell size is set as 300 and 450. The reason is that for both cases the cell size at the right border of the given virtual environment will be 100. Given this small cell size, there is a possibility that there may not be any avatar in the cell. How the layer-1cell size and AOI size affect the success rate of overlapping cell update in our collaborative IM mechanism is shown in Figure 3. The success rate of updating entity's layer-1 overlapping cells increases with increasing AOI size. This is because each entity knows more entities when the AOI size increases. The success rate also increases with the layer-1 cell size since the probability of number of avatars whose AOI overlaps with a cell border will increase when layer-1 cell size becomes larger. Developing mechanisms to tolerate failures in updating layer-1 residing cell and overlapping cells is not the focus of this paper. For the experiments in the rest part of this section, we only choose the layer-1 cell sizes which do not lead to any failure of residing cell update and overlapping cell update in the collaborative IM mechanism.

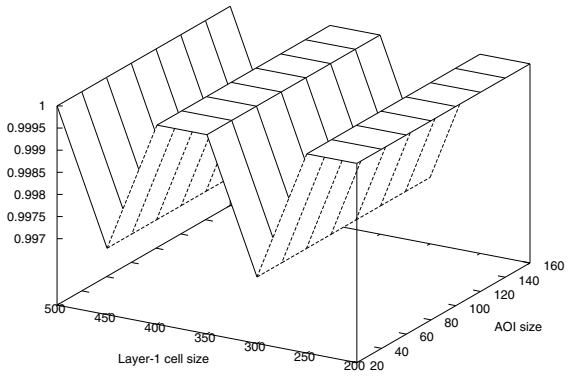


Fig. 2. Success rate of layer-1 residing cell update

C. Compare to Hybrid IM

First, we evaluated the communication cost of the collaborative IM mechanism and compared it with the hy-

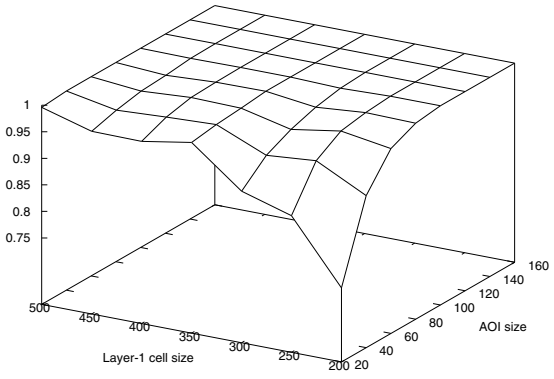


Fig. 3. Success rate of layer-1 overlapping cell update

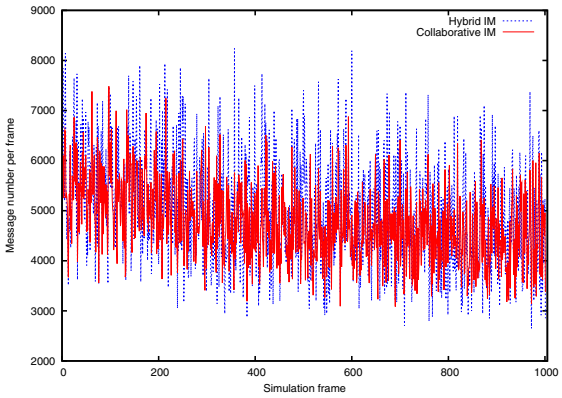


Fig. 5. Compare to hybrid IM about traffic burst

brid IM mechanism. The communication cost refers to the number of messages which are sent by all entities at each frame. The messages which are involved in updating both layer-1 residing cell and overlapping cells are classified into the *layer-1 maintenance cost*. These include the messages with type *L1RC_UPDATE*, *RC_LEAVE*, *L1RC_UPDATE_REPLY*, *OC_UPDATE* and *OC_LEAVE* in the algorithm. The status updates in the experiment only contain entity's position updates. Since the collaborative IM mechanism adopts expanded AOI approach to achieve physically correct filtering as we mentioned in Section III-C, each entity may receive the position updates from other entities which are actually outside its real AOI, and the position updates of this kind are classified as irrelevant position updates in our experiments. Similarly, relevant position updates are defined as position updates that each entity really needs.

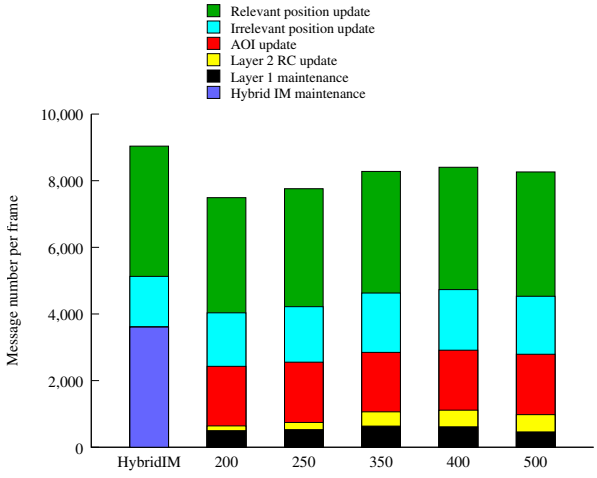


Fig. 4. Compare to hybrid IM with different layer-1 cell size

In the first experiment, entity's AOI size is 100 distance units. Recall that the performance of the hybrid IM mechanism depends on both the cell size and the distance threshold chosen [31], we choose the optimal configuration for the hybrid IM mechanism (i.e., with cell size equal to 150 distance units and space threshold equal to 9 distance units). The left-most bar in Figure 4 shows the corresponding communication cost.

The rest four bars in Figure 4 represent the communication cost in the proposed collaborative IM mechanism with different layer-1 cell size. The layer-1 cell size is set to be 200, 250, 350, 400 and 500 distance units to achieve 100 percent success rate of collaborative IM mechanism. Since the hybrid IM mechanism is adopted in the layer 2 of collaborative IM mechanism, layer-2 cell size in our collaborative IM mechanism is set as 150 distance units (which is the same as the cell size used in the hybrid IM mechanism). Figure 4 shows that the maintenance cost of our collaborative IM mechanism (i.e., the cost for layer-1 maintenance, layer-2 RC update, and AOI update) is significantly less than that of the hybrid IM mechanism. The communication cost of layer-2 residing cell update in our collaborative IM mechanism increases with increasing layer-1 cell size. Although Figure 4 does not present how the layer-1 maintenance cost is affected by different layer-1 cell size, there is an underlying trade-off that has an influence on the layer-1 maintenance cost. When the layer-1 cell size becomes larger, the situations that an entity changes its layer-1 residing cell or overlapping cells occur less frequently. However, once the situations happen, the entity needs to disseminate the messages about its new layer-1 residing cell or overlapping cells to more entities.

We also measured the number of messages at each frame during the last 1000 simulation frames and the results are shown in Figure 5. The layer-1 cell size in the collaborative IM mechanism is 400 distance units which represents the worst case of the collaborative IM mechanism according to Figure 4. As for the hybrid IM mechanism, the same settings as those in Figure 4 are used. In the hybrid IM mechanism, there are 101 simulation frames with the number of messages over 6500. Only 18 simulation frames with 6500 messages are registered in the collaborative IM mechanism. So, the collaborative IM mechanism also improves over the hybrid IM mechanism in terms of message burst.

More experiments were conducted to compare hybrid IM

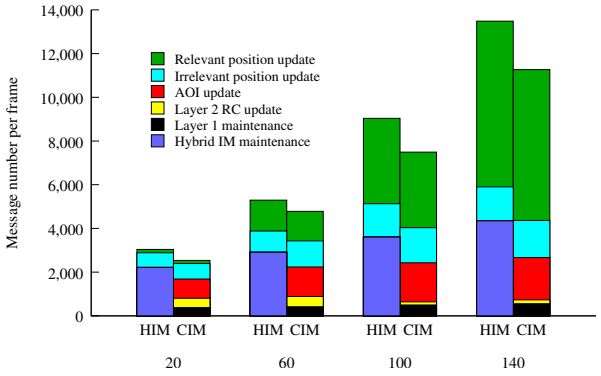


Fig. 6. Compare to hybrid IM with different AOI size

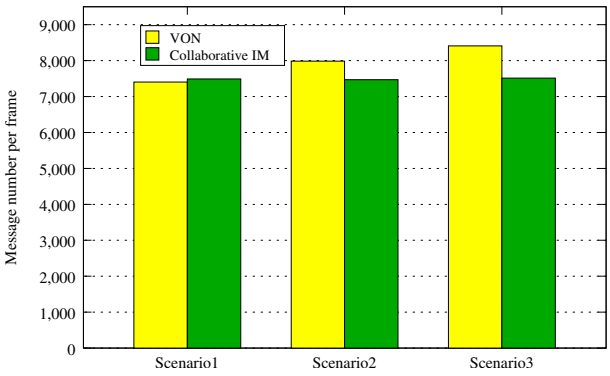


Fig. 7. Compare to VON

mechanism and our collaborative IM mechanism under different AOI settings. The corresponding results are shown in Figure 6 and for a given AOI size only those parameter values resulting in the best communication performance are given in the figure. The AOI size is selected from 20, 60, 100 and 140 distance units. For each AOI size, the left and right bars represent the communication cost of hybrid IM and collaborative IM mechanism respectively. It is easy to see that our collaborative IM mechanism always performs better than the hybrid IM mechanism.

D. Compare to VON

Another experiment was set up to compare the collaborative IM mechanism with a well-known P2P neighbor discovery mechanism called VON [18]. As mentioned in Section II-B, VON makes use of Voronoi diagrams to define the neighborship among entities. In VON, an entity keeps sending its position updates to other entities which are determined as the sending entity's AOI neighbors or enclosing neighbors at each frame. Some of those neighbors, which are classified as boundary neighbors, are responsible for informing the sending entity about some other entities that become the sending entity's new AOI neighbors or enclosing neighbors.

In order to compare the proposed collaborative IM mechanism with VON, three game scenarios were designed in the experiment. In the first scenario, all the entities have the same AOI size of 100 distance units, and the second scenario makes half of the entities change their AOI sizes to 95 distance units and the rest of them have AOI size of 105 distance units. In the third scenario, half of the entities have AOI size of 90 distance units and the rest entities have AOI size of 110 distance units. The optimal configuration for the first scenario was presented in the previous section. The layer-1 cell size, layer-2 cell size and space threshold in the optimal configuration for the rest two scenarios are 250, 100 and 10 respectively.

Figure 7 shows the communication cost of collaborative IM mechanism and VON under three game scenarios. The left and right bars for each scenario represent the communication cost of VON and collaborative IM mechanism respectively. When all entities have the same AOI size, VON performs slightly better than our collaborative IM mechanism. When entities have different AOI size, the collaborative IM mechanism has better performance than VON. Figure 7 also shows that the communication cost in VON increases when the diversity of entities' AOIs increases, and the performance advantage of the collaborative IM mechanism becomes more and more obvious.

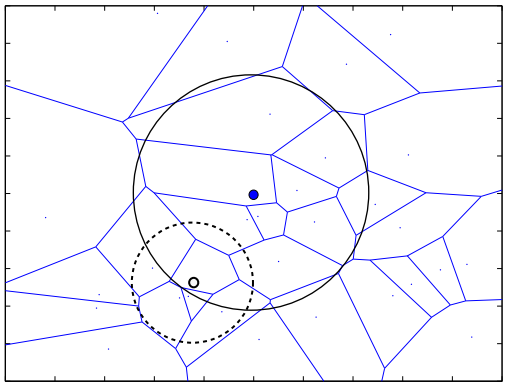


Fig. 8. VON

How does the diversity of entities' AOIs affect the communication cost in VON? We find that when two entities' AOI sizes are different, one entity may keep sending its position updates to another entity at each frame, but these position updates are actually used to maintain the neighborship among entities. For example in Figure 8, the entity, whose AOI is represented as a solid circle, sends its position update to another entity whose AOI is represented as a dashed circle. The entity whose AOI is the dashed circle actually doesn't need the position update, but it will use this position update to calculate the sending entity's new AOI. However, in the proposed collaborative IM mechanism, it is not necessary to send AOI updates to another entity which is inside the local entity's AOI at each frame (See condition 1 in Section III-A).

V. Conclusions and Future Work

In this paper, we have proposed a collaborative IM mechanism for P2P NVEs. In this mechanism, the virtual environment is partitioned into two layers of cells for describing the relationship between an entity and an AOI. The hybrid IM mechanism is used in the second layer to reduce the

communication cost of AOI updates and entities update their own information of the first layer cells in a collaborative manner. We have presented the local data structure kept by each entity and a distributed algorithm to maintain the data structure at each entity side. When an entity updates its residing cell or overlapping cells in the first layer, its local data structures are modified with the help of other entities it has already known. Compared with the hybrid IM mechanism, the proposed mechanism significantly reduces the communication cost. The experimental results also show that the proposed mechanism can achieve comparable or better performance than the well-known P2P neighbor discovery mechanism VON.

In the future work, we plan to implement our collaborative IM mechanism in a real P2P environment and evaluate its performance. We will study the performance of our collaborative IM mechanism under realistic communication pattern among players as well. The mechanisms to tolerate our algorithm failures in updating layer-1 residing cell and overlapping cells is another topic that we will look into in the future.

REFERENCES

[1] S. Singhal and M. Zyda, *Networked Virtual Environments: Design and Implementation*. ACM Press, 1999.

[2] D. Miller and J.A.Thorpe, "Simnet: the advent of simulator networking," in *Proceeding of the IEEE, vol. 83, no 8*, 1995, pp. 1114–1123.

[3] "Quake," *http://www.idsoftware.com*, id Software, 1996.

[4] "Starcraft," *http://www.blizzard.com*, Blizzard Entertainment, 1998.

[5] M. Olausson, *Online Games: Global Market Forecast*. Strategy Analytics, 2007.

[6] G. Dolbier, *Hardware and Hosting - What every online game developer should know*. IGDA Online Games Quarterly, 2005.

[7] N. E. Baughman, M. Liberatore, and B. N. Levine, "Cheat-Proof Playout for Centralized and Peer-to-Peer Gaming," *IEEE/ACM Transactions on Networking*, pp. 1–13, February 2007. [Online]. Available: http://prisms.cs.umass.edu/brian/pubs/baughman.ToN.pdf

[8] A. Shaikh, S. Sahu, M. Rosu, M. Shea, and D. Saha, "Implementation of a service platform for online games," in *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*. New York, NY, USA: ACM, 2004, pp. 106–110.

[9] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool, "The effects of loss and latency on user performance in unreal tournament 2003," in *Proceedings of ACM Network and System Support for Games Workshop (NetGames)*, 2004, pp. 144–151.

[10] SentiAtHome, "The senti@home project website," *http://setiathome.berkeley.edu/*, 2003.

[11] A. Y. Halevy, Z. G. Ives, P. Mork, and I. Tatarinov, "Piazza: data management infrastructure for semantic web applications," in *WWW*, 2003, pp. 556–567.

[12] Napster, "The napster website," *http://www.napster.com/*, 2000.

[13] Gnutella, "The gnutella website," *http://www.gnutella.com/*, 2003.

[14] J. Kubiatowicz, D. Bindel, Y. Chen, S. E. Czerwinski, P. R. Eaton, D. Geels, R. Gummadi, S. C. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Y. Zhao, "Oceanstore: An architecture for global-scale persistent storage," in *ASPLOS*, 2000, pp. 190–201.

[15] B. Knutsson, H. Lu, W. Xu, and B. Hopkins, "Peer-to-peer support for massively multiplayer games," in *Proceedings of IEEE INFOCOM*, 2004, pp. 96–107.

[16] A. P. Yu and S. T. Vuong, "Mopar: a mobile peer-to-peer overlay rrchitecture for interest management of massively multiplayer online games," in *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM, 2005, pp. 99–104.

[17] A. Bharambe, J. Pang, and S. Seshan, "Colyseus: A distributed architecture for online multiplayer games," in *Proceedings of Networked Systems Design and Implementation*, 2006, pp. 155–168.

[18] S. Y. Hu, J. F. Chen, and T. H. Chen, "Von: A scalable peer-to-peer network for virtual environments," *IEEE Network*, vol. 20, no. 4, pp. 22–31, 2006.

[19] L. Pantel and L. C. Wolf, "On the suitability of dead reckoning schemes for games," in *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*. New York, NY, USA: ACM, 2002, pp. 79–84.

[20] S. Aggarwal, H. Banavar, A. Khandelwal, S. Mukherjee, and S. Rangarajan, "Accuracy in dead-reckoning based distributed multi-player games," in *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*. New York, NY, USA: ACM, 2004, pp. 161–165.

[21] M. A. Bassiouni, M.-H. Chiu, M. Loper, M. Garnsey, and J. Williams, "Performance and reliability analysis of relevance filtering for scalable distributed interactive simulation," *ACM Trans. Model. Comput. Simul.*, vol. 7, no. 3, pp. 293–331, 1997.

[22] K. L. Morse, L. Bic, and M. Dillencourt, "Interest management in large-scale virtual environments," *Presence: Teleoper. Virtual Environ.*, vol. 9, no. 1, pp. 52–68, 2000.

[23] B. Logan and G. Theodoropoulos, "The distributed simulation of multi-agent systems," in *Proceedings of the IEEE*, 2000, pp. 174–185.

[24] G. Vulov, T. He, and M. Hybinette, "Quantitative assessment of an agent-based simulation on a time warp executive," in *Proceedings of the 40th Conference on Winter Simulation*, ser. WSC '08. Winter Simulation Conference, 2008, pp. 1068–1076.

[25] T. He and M. Hybinette, "A comparison of interest manager mechanisms for agent-based simulations using a time warp executive," in *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation*, ser. PADS '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 157–162.

[26] A. Soheili, V. Kalogeraki, and D. Gunopulos, "Spatial queries in sensor networks," in *Proceedings of the 13th annual ACM international workshop on Geographic information systems*, ser. GIS '05. New York, NY, USA: ACM, 2005, pp. 61–70.

[27] M. R. Macedonia, M. J. Zyda, D. R. Pratt, D. P. Brutzman, and P. T. Barham, "Exploiting reality with multicast groups," *IEEE Comput. Graph. Appl.*, vol. 15, no. 5, pp. 38–45, 1995.

[28] S. R. Daniel and D. J. V. Hook, "Evaluation of grid-based relevance filtering for multicast group assignment," in *Proceedings of the Distributed Interactive Simulation*, 1996, pp. 739–747.

[29] D. V. Hook and J. Calvin, "Data distribution management in rti 1.3," in *Proceedings of the 1998 Spring Simulation Interoperability Workshop*, 1998.

[30] D. D. Wood, "Implementation of ddm in the mak high performance rti," in *Proceedings of the 2002 Spring Simulation Interoperability Workshop*, 2002.

[31] K. Pan, W. Cai, X. Tang, S. Zhou, and S. Turner, "A hybrid interest management mechanism for peer-to-peer networked virtual environments," in *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2010, pp. 1 – 12.

[32] R. Ayani, F. Moradi, and G. Tan, "Optimizing cell-size in grid-based ddm," in *PADS '00: Proceedings of the fourteenth workshop on Parallel and distributed simulation*. Washington, DC, USA: IEEE Computer Society, 2000, pp. 93–100.

[33] K. L. Morse and J. S. Steinman, "Data distribution management in the hla: Multidimensional regions and physically correct filtering," in *Proceedings of the 1997 Spring Simulation Interoperability Workshop*. Spring, 1997, pp. 343–352.

[34] S. Rieche, K. Wehrle, M. Fouquet, H. Niedermayer, L. Petrak, and G. Carle, "Peer-to-peer-based infrastructure support for massively multiplayer online games," in *Proceedings of 4th Annual IEEE Consumer Communications and Networking Conference (CCNC 2007)*, 2007.

[35] J. Keller and G. Simon, "Solipsis: A massively multi-participant virtual world," in *PDPTA*, H. R. Arabnia and Y. Mun, Eds. CSREA Press, 2003, pp. 262–268.

[36] M. Garetto and E. Leonardi, "Analysis of random mobility models with partial differential equations," *IEEE Trans. Mob. Comput.*, vol. 6, no. 11, pp. 1204–1217, 2007.

[37] H. Liang, R. N. Silva, W. T. Ooi, and M. Motani, "Avatar mobility in user-created networked virtual worlds: measurements, analysis, and implications," *Multimedia Tools Appl.*, vol. 45, pp. 163–190, October 2009.

Dead Reckoning-Based Update Scheduling Against Message Loss for Improving Consistency in DVEs

Zengxiang Li, Wentong Cai, Xueyan Tang
Parallel and Distributed Computing Center
School of Computing Engineering
Nanyang Technological University, Singapore 639798
Email: {zxli, aswtcai, asxytang}@ntu.edu.sg

Suiping Zhou
School of Computing
Teesside University, UK
Email: S.Zhou@tees.ac.uk

***Abstract*—Maintaining a consistent presentation of the virtual world among participants is a fundamental problem in the Distributed Virtual Environment (DVE). The problem is exacerbated due to the limited network bandwidth and error-prone transmission. This paper investigates Dead Reckoning (DR) update scheduling to improve consistency in the DVE against message loss. Using the metric of Time-Space Inconsistency (TSI), which combines the spatial magnitude and temporal duration of inconsistency, we analytically derive the impact of message loss on TSI when using a DR-based update mechanism. To improve consistency against message loss, a naive update scheduling algorithm is first proposed, in which the expected spatial difference is calculated by taking message loss into account. In order to reduce the TSI to the case without transmission failures, a compensation update scheduling algorithm is further proposed by reducing the DR threshold according to the message loss rate. Using these algorithms, a budget-based mechanism is developed to meet the network bandwidth constraint. We show through experiments using a racing car game that the budget-based mechanism using the compensation update scheduling algorithm makes the best use of available network bandwidth to reduce the inconsistency and its impact on the participants. In addition, it ensures fairness among participants in spite of widely varying message loss rates.**

I. Introduction

A distributed virtual environment (DVE) encourages a large amount of participants from different corners of the world to communicate and interact with each other in a virtual world [1]. DVEs are widely used in various areas such as military training [2], e-learning [3], and massively multiplayer online games [4]. Maintaining a consistent presentation of the virtual world among participants is a fundamental problem in the DVE. Once the state of an entity in the virtual world changes, all participants should be able to observe the change in real time. Otherwise, inconsistent views can seriously affect meaningful interactions among the participants. For instance, some players in a car racing game may behave unexpectedly after observing the wrong positions of their cars. Recently, wireless distributed virtual environments have attracted more and more interests, because wireless networks, especially IEEE 802.11 WLAN [5], have emerged as a common last-hop in the Internet. However, wireless networks commonly have limited bandwidth due to the power restriction. They are also error-prone due to various factors such as obstacles and signal interference. For these reasons, the problem of consistency maintenance in the DVE is exacerbated.

Dead Reckoning (DR) [6], [7] is commonly used to extrapolate the position of an entity, for reducing the amount of data exchanged over the network. In this mechanism, DR updates of the entity are sent only when the difference between real position and extrapolated position is greater than a predefined threshold. In this paper, we investigate DR-based update scheduling to utilize the limited network bandwidth efficiently and to improve consistency in DVEs in the presence of transmission failures. Zhou et al [8] have proposed a metric called *Time-Space Inconsistency* (TSI) to measure the inconsistency by combining its spatial magnitude and temporal duration. It has been shown that TSI is inherent in DVEs and may significantly affect the perceptions of participants.

Due to network failures, some DR updates might be lost in the transmission. As a result, the receiver would fail to use the most recently sent DR update to extrapolate the position of the entity, which would increase the inconsistency of the entity. Using the metric of TSI, we analytically derive the impact of message loss on inconsistency when using a DR-based update mechanism. To improve consistency against message loss, a naive update scheduling algorithm is first proposed. In this algorithm, message loss in the transmission is taken into account to calculate the expected spatial difference between the real position of an entity and its extrapolated position at the receiver. A DR update is sent when the expected spatial difference exceeds the predefined threshold. In order to reduce the TSI to the case without transmission failures, a compensation update scheduling algorithm is further proposed by reducing the DR threshold according to the message loss rate. With limited bandwidth provided by the network, some DR updates may not be sent in time. To send DR updates selectively within the bandwidth constraint, a budget-based mechanism is then developed using the above update scheduling algorithms. Experimental results show that the budget-based mechanism using the compensation algorithm has advantages over that using the naive algorithm. It can provide fairness among participants in spite of widely varying transmission failures. With enough network bandwidth available, it can even reduce the inconsistency and its impact on the perceptions of participants to the level of the case without transmission failures.

The rest of the paper is organized as follows: Section II discusses the related work. Section III introduces the system

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

model and our objectives. Section IV presents the naive and compensation update scheduling algorithms, as well as the budget-based mechanism. Section V describes the experiments using a racing car game and discusses the experimental results. Finally, Section VI concludes the paper and outlines the future work.

II. Related work

In Dead Reckoning (DR) [6], [7], each participant maintains a DR model to extrapolate the position of a moving entity between updates. Hence, participants can observe the approximate view using less number of position updates. In a DR-based update mechanism, there is inconsistency between the real position and the extrapolated position of an entity. It is straight-forward to measure the inconsistency using the spatial difference. Zhou et al [8] have verified that the temporal duration of the inconsistency also affects human perception significantly. Therefore, the metric TSI is proposed to measure inconsistency by combining spatial difference and temporal duration. Similarly, the cumulated error within the update interval is used to measure the inconsistency in [9], [10]. Roberts et al [11] have also proposed to use an alternative DR threshold based on spatial difference and temporal duration.

As shown in [8], [12], the sender and receiver of DR updates may observe inconsistent views of an entity, if their clocks are not synchronized. A globally synchronized clock can be used to reduce the inconsistency of the entity. The impact of message delay on the inconsistency of entities and on the player's experience in the virtual world is studied in [8], [12], [13], [14], [15]. These studies have reached an agreement that the inconsistency increases with increasing network delays. Local-lag technique is used in [16] to account for network delays. Different from the above work, we focus on improving consistency in DVEs in the presence of transmission failures in the DR-based update mechanism. As illustrated in [15], players in a car racing game may observe different positional relations of cars due to message loss. Cheung and Sakamoto [17] have proposed to use application-aware redundant Extrapolated Parity Packets to recover the lost DR updates. Instead of recovering the lost DR updates which might be out of date anyway, we propose to schedule DR updates more frequently to reduce the impact of message loss on the TSI of entities.

With the constraint of network bandwidth, the sender may fail to send DR updates to all receivers in time. To utilize the limited bandwidth efficiently, a number of update scheduling algorithms (e.g., [18], [19], [20], [21]) have been proposed. Different from these algorithms, the update scheduling algorithms proposed in this paper take message loss into account. A budget-based mechanism is further developed using the proposed update scheduling algorithms to send DR updates selectively within the bandwidth constraint.

III. System Model

A DVE normally consists of a group of interconnected and geographically distributed computers (nodes) that simulate the entities in the virtual world. The nodes are generally organized in either a client-server or a peer-to-peer architecture. Without loss of generality, this paper considers update dissemination from a node hosting an entity to another node interested in the state of the entity. The former is called the sender, which can be either a server in the client-server architecture or a peer in the peer-to-peer architecture. The latter is called the receiver, which is either a client in the client-server architecture or a peer in the peer-to-peer architecture. In this paper, we focus on the updates of entity positions in the virtual world.

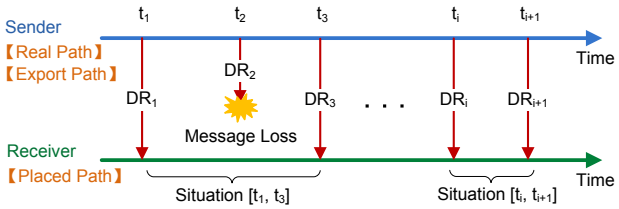


Fig. 1. System model of a DR-based update mechanism

The system model of a DR-based update mechanism is shown in Figure 1, where a sender sends DR updates to a receiver. The sender refreshes the entity position in real time according to user actions, e.g., accelerating, braking and changing orientation in a racing game. Following the terminologies defined by Aggarwal et al [12], the trajectory of the entity at the sender side is referred to as the *real path*. The position of the entity in its real path at time t is denoted as $\mathcal{R}(t)$. Besides that real path, the sender also keeps the *exported path*, i.e., the path extrapolated using previously sent DR updates. The position of the entity in its exported path at time t is denoted as $\mathcal{E}(t)$. The difference between real path and the exported path is referred to as the *DR error*. The DR error at time t denoted as $DE(t)$ is equal to $\mathcal{R}(t) - \mathcal{E}(t)$. Using the received DR updates, the receiver renders the trajectory of the entity, which is referred to as the *placed path*. The position of the entity in its placed path at time t is denoted as $\mathcal{P}(t)$. A deviation between the exported path and the placed path of the entity is referred to as the *export error*. The export error at time t denoted as $EE(t)$ is equal to $\mathcal{E}(t) - \mathcal{P}(t)$. The export error exists due to the clock asynchrony [8], [12], message delay [8], [14], [12], or message loss in the transmission [15]. The *real error* is defined as the inconsistency between the real path of the entity at the sender and the placed path at the receiver. The real error at time t denoted as $RE(t)$ is equal to $\mathcal{R}(t) - \mathcal{P}(t)$. It is composed of the DR error and the export error.

In this paper, we propose new DR-based update scheduling algorithms to reduce the real error of the entity in the presence of message loss. As mentioned in Section II, the impact of clock asynchrony and message delay can be handled by a global synchronized clock (e.g., [8], [12]) and local lag technique (e.g., [16]) respectively. For simplicity, we assume the clocks of the sender and receiver are synchronized and the message delay in the transmission can be ignored.

The TSI metric [8] is used in this paper to measure the inconsistency of the entity. On receiving the DR updates, the position of the entity is updated at the receiver. Thus, the real error is brought down to 0 at times of update receipts. Between updates receipts, the real error can be greater than 0. The time interval between two successive DR update receipts is referred to as a situation. Let t_b and t_e be the times of two successive update receipts. Then, the TSI of the situation [t_b, t_e] is calculated as:

$$\int_{t_b}^{t_e} RE(t)dt$$

Due to the constraint of network bandwidth, a sender can send DR updates up to only a certain number to its receivers at any given time. We assume that the underlying network supports unicast transmission only. Without enough bandwidth, the sender may fail to send the DR updates in time. In addition, the message loss rates in the transmission between the sender and receivers (for short, the message loss rates of receivers) may vary widely. Due to different message loss rates, the receivers may observe different levels of inconsistency for the same entity even if the same set of DR updates are sent by the sender to various receivers. Therefore, the sender has to decide which DR updates to send to which receivers to reduce the differences of inconsistency amongst receivers within the bandwidth constraint. In this paper, we investigate how to schedule DR updates for the following two objectives:

- $\mathcal{O}1$: obtain equal TSIs of an entity between the sender and various receivers irrespective of their different message loss rates.
- $\mathcal{O}2$: reduce the TSIs of an entity to the level of the case without message loss.

As shown in [8], the impact of inconsistency on the perceptions of participants is highly dependent on its TSI. Once the above objectives have been achieved, participants in the virtual world can compete fairly and enjoy the same experience as that without transmission failures.

IV. DR-based Update Scheduling

A. Impact of Message Loss

Consider the scenario that there is no bandwidth constraint for a sender to send DR updates of an entity to a receiver. First, we analytically derive impact of message loss on the TSI of the entity when using the original update scheduling algorithm. For simplicity, we shall assume one dimensional linear movements in our analysis. However, the analysis and algorithms presented in this paper can also be extended to multiple dimensional movements.

The real path of the entity can be modeled using Taylor series expansion approximately, because $\mathcal{R}(t)$ is assumed to be a continuous and differentiable function of the variable t [10]. Suppose that, a DR update DR_i of an entity is sent at time t_i. The real path of the entity right after t_i can be estimated using following equation, where v_i and a_i denote the velocity and acceleration of the entity at time t_i:

$$\mathcal{R}(t_i + x) \approx \mathcal{R}(t_i) + \frac{v_i}{1!}x + \frac{a_i}{2!} \cdot x^2$$

We assume that first order extrapolation formula is used in the DR-based update mechanism. Then, the exported path of the entity is given by:

$$\mathcal{E}(t_i + x) = \mathcal{R}(t_i) + v_i \cdot x$$

Therefore, the DR error is:

$$DE(t_i + x) = \mathcal{R}(t_i + x) - \mathcal{E}(t_i + x) \approx \frac{1}{2} \cdot a_i \cdot x^2$$

As mentioned in Section III, the sender maintains the real path and exported path of the entity. Hence, it can calculate the DR error at any time. In the original update scheduling algorithm, a DR update is sent once the DR error exceeds a predefined threshold δ. That is, the next DR update DR_{i+1} should be sent at time t_{i+1}, when:

$$DE(t_{i+1}) \approx \frac{1}{2} \cdot a_i \cdot (t_{i+1} - t_i)^2 = \delta$$

Thus, the duration between the two successively sent DR updates DR_i and DR_{i+1}, which is denoted as λ_i, can be calculated as:

$$\lambda_i = t_{i+1} - t_i = \sqrt{\frac{2\delta}{a_i}} \tag{1}$$

In the case that no DR update is lost, both DR_i and DR_{i+1} are received by the receiver. So, for any $0 \leq x \leq \lambda_i$, $RE(t_i + x) = DE(t_i + x)$. Thus, the TSI of the situation [t_i, t_{i+1}] is given by:

$$\Omega_i = \int_0^{\lambda_i} \frac{1}{2} \cdot a_i \cdot x^2 \, dx = \frac{1}{6} \cdot a_i \cdot \lambda_i^3 = \sqrt{\frac{2\delta^3}{9 \cdot a_i}} \tag{2}$$

As shown in Equations (1) and (2), the smaller the acceleration, the greater the length and the TSI of the situation. However, the time-averaged TSI of the situation, i.e., the ratio between TSI and length of the situation,

$$\frac{\Omega_i}{\lambda_i} = \frac{\delta}{3}$$

is only concerned with the threshold. It is not concerned with entity's kinetics, such as velocity and acceleration. Suppose that the threshold is a constant value over a given time period T, all situations during the period should have the same time-averaged TSIs. Therefore, the total TSI during the period can be calculated as follows:

$$\sum \Omega = \frac{\delta}{3} \cdot T \tag{3}$$

In the case that some DR updates are lost due to transmission failures, the real error of an entity generally increases as the receiver may not be able to use the most recently sent DR update to extrapolate the entity's position. In the case that updates DR_{i+1}, DR_{i+2}, $\cdots$, and DR_{i+k-1} are all lost, the two successively received DR updates are DR_i and DR_{i+k}.

Suppose that the threshold and acceleration of the entity is a constant value during the situation $[t_i,\ t_{i+k}]$, it follows from Equation 1 that the durations between two successively sent DR updates are all the same during the situation, i.e., $\lambda_i = \lambda_{i+1} = \cdots = \lambda_{i+k-1}$. Hence, the length of the situation $[t_i, t_{i+k}]$ is $k \cdot \lambda_i$. We can also get the placed path of the entity during the situation as follows:

$$\mathcal{P}(t_i + x) = \mathcal{R}(t_i) + v_i \cdot x$$

for any $0 \leq x \leq k \cdot \lambda_i$. Therefore,

$$RE(t_i + x) = \mathcal{R}(t_i + x) - \mathcal{P}(t_i + x) = \frac{1}{2} \cdot a_i \cdot x^2$$

and the TSI of the situation $[t_i,\ t_{i+k}]$ is:

$$\int_0^{k \cdot \lambda_i} \frac{1}{2} \cdot a_i \cdot x^2 \, dx = \frac{1}{6} \cdot a_i \cdot (k \cdot \lambda_i)^3 = k^3 \cdot \Omega_i$$

As described above, the situation $[t_i, t_{i+k}]$ occurs if DR_{i+1}, DR_{i+2}, $\cdots$, and DR_{i+k-1} are all lost and DR_i and DR_{i+k} are received. Assuming that the message loss rate is a constant value p, thus, the probability of the occurrence of situation $[t_i, t_{i+k}]$ is thus $p^{k-1} \cdot (1-p)$. Hence, the expected length and the expected TSI of the situation starting from t_i can be calculated as follows (refer to Appendix for the derivation of Equation 5):

$$\sum_{k=1}^{\infty} p^{k-1} \cdot (1-p) \cdot k \cdot \lambda_i = \frac{\lambda_i}{1-p} \tag{4}$$

$$\sum_{k=1}^{\infty} p^{k-1} \cdot (1-p) \cdot k^3 \cdot \Omega_i = \frac{1 + 4p + p^2}{(1-p)^3} \cdot \Omega_i \tag{5}$$

From the above two equations, we can then obtain the time-averaged TSI of the situation as follows:

$$\frac{\frac{1+4p+p^2}{(1-p)^3} \cdot \Omega_i}{\frac{\lambda_i}{1-p}} = \frac{1 + 4p + p^2}{(1-p)^2} \cdot \frac{\Omega_i}{\lambda_i} = \frac{1 + 4p + p^2}{(1-p)^2} \cdot \frac{\delta}{3}$$

Similar to the case without message loss, we can infer that the time-averaged TSI is only concerned with the threshold and message loss rate, which are both assumed to be constant values. Therefore, the total TSI over a given time period T can be calculated as:

$$\sum \Omega = \frac{1 + 4p + p^2}{(1-p)^2} \cdot \frac{\delta}{3} \cdot T \tag{6}$$

To illustrate the impact of message loss on inconsistency, the total TSI in the case of message loss is normalized by that in the case without message loss (Equation (3)) and then plotted as the "Original" curve in Figure 2. As can be seen, using the original update scheduling algorithm, the normalized total TSI increases sharply with increasing message loss rate. For this reason, new update scheduling algorithms are required to reduce the TSI in the presence of message loss.

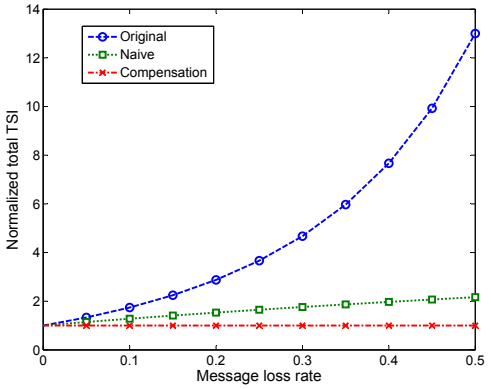


Fig. 2. Normalized total TSI versus message loss rate

B. Naive Algorithm

In the original update scheduling algorithm, a DR update is sent once the DR error exceeds the threshold. However, due to message loss, the real error, which is composed of the DR error and the export error is generally greater than the threshold at the time of DR update. Thus, an intuitive strategy to reduce inconsistency is to send the DR update once the real error exceeds the threshold. Unfortunately, the sender, which does not know the placed path of the entity, cannot calculate the exact value of the real error. Instead, the expected real error can be calculated based on previously sent DR updates and the message loss rate. Suppose that DR_i is the last sent DR update by the sender and DR_{i-j+1}, where $j \geq 1$, is the last received DR update at the receiver. Then, the receiver has to extrapolate the entity's placed path at time $t_i + x$ (the time right after t_i) using DR_{i-j+1}. Assume $RE(t, DR_k)$ denotes the real error at time t given that the last received DR update at the receiver is DR_k $(t > t_k)$. In the above case, $j-1$ successively sent DR updates, DR_{i-j+2}, $\cdots$, DR_{i-1} and DR_i, are lost. The probability for this case to occur is $p^{j-1} \cdot (1-p)$. Consequently, the expected $RE(t_i + x)$ can be calculated as follows:

$$\sum_{j=1}^{\infty} p^{j-1} \cdot (1-p) \cdot RE(t_i + x, DR_{i-j+1}) \tag{7}$$

Therefore, the next DR update DR_{i+1} should be sent at time t_{i+1} when:

$$\sum_{j=1}^{\infty} p^{j-1} \cdot (1-p) \cdot RE(t_{i+1}, DR_{i-j+1}) = \delta$$

We shall refer this algorithm as the naive update scheduling algorithm.

The duration between the two successively sent DR updates DR_i and DR_{i+1} in this algorithm is denoted as λ'_i, which is equals to $t_{i+1} - t_i$. In the case that the last received DR update at the receiver is DR_i, the real error at t_{i+1} equals the DR error at that time and is calculated as:

$$RE(t_{i+1}, DR_i) = DE(t_{i+1}) = DE(t_i + \lambda'_i) = \frac{1}{2} \cdot a_i \cdot \lambda'^2_i \tag{8}$$

In the case that the last received DR update is DR_{i-j+1}, the real error at time t_{i+1} is generally greater than the DR error at that time. Suppose that the message loss rate and acceleration are constant values during the period between t_{i-j+1} and t_{i+1}. The durations between any two successively sent DR updates are the same, i.e., $\lambda'_{i-j+1} = \cdots = \lambda'_{i-1} = \lambda'_i$. So, $t_{i+1} - t_{i-j+1} = j \cdot \lambda'_i$. Consequently, the real error at time t_{i+1} can be calculated as:

$$RE(t_{i+1}, DR_{i-j+1}) = \frac{1}{2} \cdot a_{i-j+1} \cdot (j \cdot \lambda'_i)^2 = \frac{1}{2} \cdot a_i \cdot (j \cdot \lambda'_i)^2$$

The probability for this case to occur, as mentioned above, is $p^{j-1} \cdot (1-p)$. Hence, the expected $RE(t_{i+1})$ can be calculated as (refer to Appendix for the derivation of Equation 9):

$$\sum_{j=1}^{\infty} p^{j-1} \cdot (1-p) \cdot \frac{1}{2} \cdot a_i \cdot (j \cdot \lambda'_i)^2 = \frac{1+p}{(1-p)^2} \cdot \frac{1}{2} \cdot a_i \cdot \lambda'^2_i \quad (9)$$

In the naive update scheduling algorithm, the next DR update DR_{i+1} is sent when the expected $RE(t_{i+1}) = \delta$. Hence, λ'_i can be calculated as:

$$\lambda'_i = \frac{1-p}{\sqrt{1+p}} \cdot \sqrt{\frac{2\delta}{a_i}} = \frac{1-p}{\sqrt{1+p}} \cdot \lambda_i \quad (10)$$

Because $0 \leq p \leq 1$, $(1-p)/\sqrt{1+p} \leq 1$, and thus $\lambda'_i \leq \lambda_i$. As a result, more DR updates are sent by the sender using the naive update scheduling algorithm than that using the original update scheduling algorithm.

If both DR_i and DR_{i+1} are received by the receiver, we can calculate the TSI of the situation $[t_i, t_{i+1}]$ denoted as Ω'_i, in a similar way as Ω_i calculated in Equation (2). That is,

$$\Omega'_i = \int_0^{\lambda'_i} \frac{1}{2} \cdot a_i \cdot x^2 \, dx = \frac{1}{6} \cdot a_i \cdot \lambda'^3_i. \quad (11)$$

Assuming that the message loss rate is a constant value p, the expected length and the expected TSI of the situation starting from t_i can also be calculated in a similar way as those calculated in Equations (4) and (5). Hence, the time-averaged TSI of the situation can be given by:

$$\frac{\frac{1+4p+p^2}{(1-p)^3} \cdot \Omega'_i}{\frac{\lambda'_i}{1-p}} = \frac{1+4p+p^2}{(1-p)^2} \cdot \frac{\Omega'_i}{\lambda'_i}$$

Using Equations (10) and (11), the above can be simplified as follows:

$$\frac{1+4p+p^2}{1+p} \cdot \frac{\delta}{3}$$

Therefore, the total TSI of the entity over a given time period T is:

$$\sum \Omega = \frac{1+4p+p^2}{1+p} \cdot \frac{\delta}{3} \cdot T \quad (12)$$

To compare with other scheduling algorithms, the above total TSI is normalized by that in the case without message loss (Equation (3)) and then plotted as the "Naive" curve in Figure 2. It can be seen that the total TSI of the naive update scheduling algorithm is much lower than that of the original update scheduling algorithm when there is message loss. However, the total TSI is still greater than that in the case of no message loss. Moreover, it still increases with increasing message loss rate.

In the implementation of the naive update scheduling algorithm, calculating the expected real error using Equation (7) directly could be time-consuming. In fact, it may not be necessary to calculate the expected real error based on all previously sent DR updates. The above theoretical analysis is also helpful to determine the number of previously sent DR updates which should be involved in the calculation of the expected real error. The value of $p^{j-1} \cdot (1-p) \cdot j^2$, a coefficients of the item in Equation (9), is negligible when $j > 16$ and $0\% \leq p \leq 50\%$. Therefore, when the message loss rate does not exceed 50%, the expected $RE(t_i + x)$ can be approximated by:

$$\sum_{j=1}^{16} p^{j-1} \cdot (1-p) \cdot RE(t_i + x, DR_{i-j+1})$$

In other words, the sender can estimate the expected real error using the past 16 DR updates sent.

C. Compensation Algorithm

As shown in Equation (6), the total TSI of the entity is determined by the message loss rate and the threshold. Hence, it is possible to reduce the TSI to the level of the case without message loss (Equation (3)) by reducing the threshold according to the message loss rate. To do so, the reduced threshold δ'' should satisfy:

$$\frac{1+4p+p^2}{(1-p)^2} \cdot \frac{\delta''}{3} \cdot T = \frac{\delta}{3} \cdot T$$

Hence, δ'' can be calculated as:

$$\delta'' = \frac{(1-p)^2}{1+4p+p^2} \cdot \delta \quad (13)$$

As a result, the duration between two successively sent DR updates DR_i and DR_{i+1} decreases to:

$$\begin{aligned} \sqrt{\frac{2\delta''}{a_i}} &= \frac{1-p}{\sqrt{1+4p+p^2}} \cdot \sqrt{\frac{2\delta}{a_i}} \\ &= \frac{1-p}{\sqrt{1+4p+p^2}} \cdot \lambda_i \end{aligned} \quad (14)$$

We shall refer to this algorithm as the compensation update scheduling algorithm. Using this algorithm, the total TSI is equal to that of the case without message loss. That is, the total TSI normalized by that of the case without message loss (Equation (3)) equals 1, as shown by the "Compensation" curve in Figure 2. Therefore, the receivers should enjoy the same experience in the virtual world regardless of whether and how frequent the messages are lost.

Compared with the naive update scheduling algorithm, the compensation algorithm is easier to implement. It is the same as the original algorithm except for reducing the threshold according to the message loss rate. Moreover, the compensation algorithm is able to reduce the TSI of the entity to that of the case without message loss and ensure the fairness among receivers with different message loss rates. However, DR updates are sent more frequently in the compensation algorithm (Equation (14)) than those in the naive algorithm (Equation (10)). In the next subsection, a budget-based mechanism is developed to deal with the network bandwidth constraint using these update scheduling algorithms.

D. Budget-based Mechanism

In a DVE, a sender is generally required to send DR updates of multiple entities to their corresponding interested receivers. Due to the constraint of network bandwidth, the sender can send DR updates up to only a certain number to the receivers at any given time. Now, we develop a budget-based mechanism to send DR updates selectively within the bandwidth constraint.

Specifically, a priority is calculated for each combination of an entity and a receiver. Suppose that DR_i is the last sent DR update of the entity to the receiver and DR_i is sent at time t_i. Using the original update scheduling algorithm, the next DR update DR_{i+1} is sent when $DE(t_i + x)$ exceeds δ. Accordingly, the priority at time $t_i + x$ is defined as:

$$\frac{DE(t_i + x)}{\delta}$$

Using the naive algorithm, the next DR update DR_{i+1} is sent when the expected $RE(t_i + x)$ exceeds δ. Accordingly, the priority at time $t_i + x$ is defined as:

$$\frac{Expected\ RE(t_i + x)}{\delta}$$

Using the compensation algorithm, the next DR update DR_{i+1} is sent when $DE(t_i+x)$ exceeds δ''. Accordingly, the priority at time $t_i + x$ is defined as:

$$\frac{DE(t_i + x)}{\delta''} = \frac{(1 + 4p + p^2)}{(1 - p)^2} \cdot \frac{DE(t_i + x)}{\delta}$$

Only when the priority is greater than 1, it is necessary to send the DR update of the entity to the corresponding receiver. Suppose that the sender can send up to W DR updates to the receivers at a time. In the case that more than W combinations of entities and receivers have priorities greater than 1, the W combinations with the highest priorities are chosen and the DR updates corresponding to these combinations are sent. After sending the updates, the priorities of these W combinations would drop to 0. Thus, the remaining combinations would be more likely to be chosen in the future. If fewer than W combinations have priorities greater than 1, only part of the network bandwidth is used to send DR updates for all these combinations with priorities greater than 1.

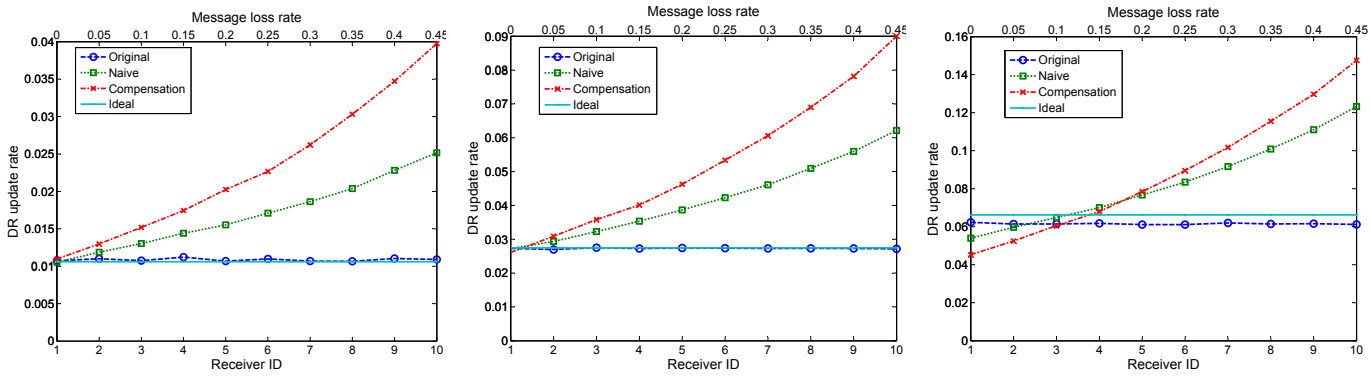
With enough bandwidth available, DR updates can be sent for all combinations with priorities greater than 1. Otherwise, DR updates are sent only for combinations with priorities greater than a value h (which is greater than 1). This has the same effect on the inconsistency of the entities as increasing the DR threshold from δ to $h \cdot \delta$. For the budget-based mechanism using the compensation update scheduling algorithm, the time-averaged TSIs of an entity at receivers with different message loss rates are all close to $\frac{\delta}{3}$ if enough bandwidth is available. Therefore, we can infer that objectives $\mathcal{O}1$ and $\mathcal{O}2$ mentioned in section III are achieved. If there is not enough bandwidth, the time-averaged TSIs of an entity at the receivers are increased to $h \cdot \frac{\delta}{3}$. Therefore, objective $\mathcal{O}1$ is still achieved. However, the objective $\mathcal{O}2$ is not achieved due to the insufficient number of DR updates sent. For the budget-based mechanisms using the original and naive update scheduling algorithms, the TSIs of an entity at the receiver with message loss rate p are close to Equations (6) and (12) respectively if enough bandwidth is available. Otherwise, the TSIs increase to h times of Equations (6) and (12) respectively. Therefore, the TSIs never reduce to the level in the case of no message loss and the receivers with higher message loss rates have greater TSIs. Hence, neither of objectives $\mathcal{O}1$ nor $\mathcal{O}2$ is achieved.

V. Experiment and Results

A. Experiment Design

In order to evaluate and compare the update scheduling algorithms proposed in this paper, simulation experiments are carried out using the traces of car positions in a car racing game called TORCS [22]. In TORCS, the position of a car is described by its X,Y and Z coordinates in the three dimensional virtual world. Since the cars can only run along a particular track, for simplicity, we model the track as one dimensional space and map the car positions into this space by calculating the distance (in meters) travelled by the car along the track. Using the calculated distance, we can also obtain the positional relations among different cars (i.e., which car is in front of another) that affect players most significantly in a racing game. There are $32,000$ recordings of car positions in each trace. The time interval between two successive position recordings in the traces is 0.02 second, which is assumed to be the length of a frame.

A simulator is developed to simulate 11 players competing in an online car racing game. The players communicate with each other in a peer to peer manner. Each player controls one car and sends DR updates of the car to all other players. The players are assumed to have synchronized clocks and the transmission delays of DR updates are assumed to be negligible. In the simulation, we focus on the DR updates from the first player to the other 10 players. The first player is referred to as the sender, whereas the other 10 players are referred to as the receivers with IDs from 1 to 10. The message loss rates of receivers 1, 2, $\cdots$, 10 are assumed to be 0%, 5%, $\cdots$, 45% respectively. The traces of two different cars collected from a TORCS game are used. One trace is used to simulate the car controlled by the sender. The other trace is used to simulate the cars controlled by the receivers. To compare the impact of message loss on consistency, the same



(a) Threshold = 5 (b) Threshold = 1 (c) Threshold = 0.2

Fig. 3. DR Update rate of receivers in different scenarios

trace is used to simulate the cars controlled by all receivers that have different message loss rates.

Four scenarios are investigated in our experiments. In the Ideal scenario, the original update scheduling algorithm is used without bandwidth constraint and message loss. In the Original, Naive and Compensation scenarios, the budget-based mechanism is applied, using the original, naive, and compensation update scheduling algorithms respectively. We assume that due to the bandwidth constraint, the sender can only send a DR update of its car to one receiver in each frame.

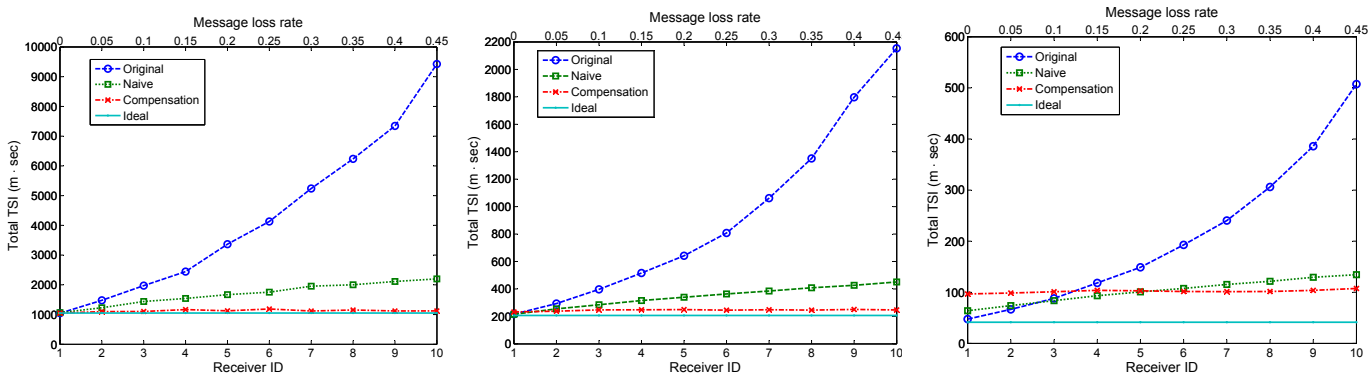
B. Experiment Results

The DR update rate of a receiver is defined as the ratio between the number of DR updates sent by the sender to the receiver and the total number of frames in the simulation. Figure 3 shows the update rates of receivers in different scenarios for different thresholds. With decreasing threshold, the priorities in all scenarios increase, and thus the numbers of DR updates sent by the sender to all receivers increase. In the Original scenario, message loss is not taken into account in update scheduling. Thus, similar numbers of DR updates are sent to all receivers just like the Ideal scenario. Different from the Original scenario, in the Compensation and Naive scenarios, the priorities of the combinations of sender-controlled car and receivers increase with increasing message loss rate. Hence, larger numbers of DR updates are sent to receivers with higher message loss rates. We can further observe from Figure 3 that this trend is more significant in the Compensation scenario. The number of DR updates sent to receivers is greater in the Compensation scenario when enough bandwidth is available, e.g., at thresholds 5 and 1 (see Figures 3(a) and 3(b)).

Figure 4 shows the total TSIs of receivers in different scenarios for different thresholds. With decreasing threshold, the number of DR updates sent to receivers increases, and thus the TSIs of the sender-controlled car at the receivers decrease. In the Ideal scenario, no DR update is lost, and thus the TSIs of different receivers are the same. The TSIs in the Original scenario increase dramatically with respect to the message loss rates of the receivers. The TSIs are reduced significantly in the Naive and Compensation scenarios due to higher DR update rates.

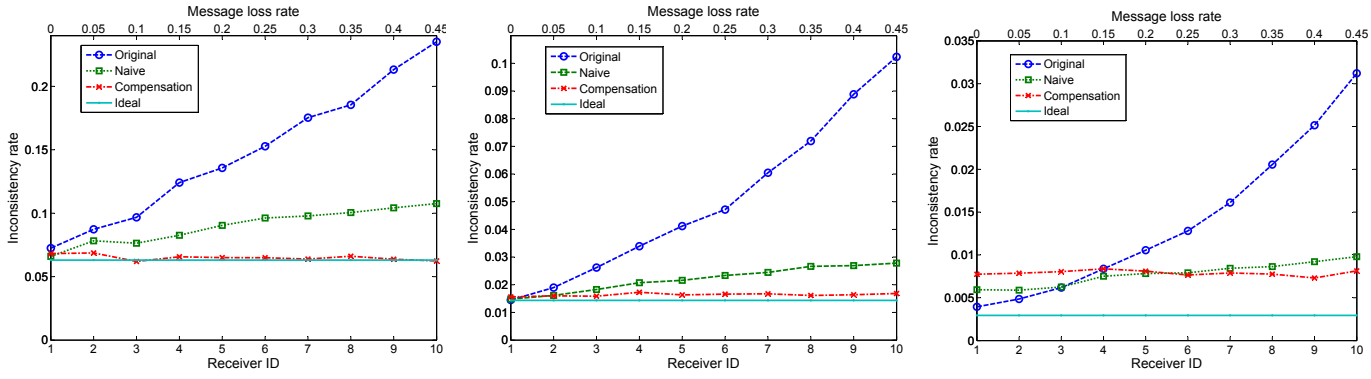
For the Compensation scenario, we can observe that the TSIs of all receivers are comparable. The TSIs are close to those in Ideal scenario when the threshold is equal to 5 or 1, but are greater than those in the Ideal scenario when the threshold is equal to *0.2*. These observations are consistent with the analysis at the end of Section IV. That is, the budget-based mechanism using the compensation update scheduling algorithm can ensure the fairness among receivers in terms of the TSIs of the sender-controlled car. With enough bandwidth available, it can reduce the TSIs to the level of the case without message loss. For the Naive scenario, we can observe from Figure 4 that the TSIs of receivers increase with increasing message loss rates and never reduce to the levels in the Ideal scenario. These observations also agree with the analysis at the end of Section IV. That is, the budget-based mechanism using the naive update scheduling algorithm can neither ensure fairness among receivers nor reduce the TSIs to the level of the case without message loss.

When the threshold is equal to *0.2*, the bandwidth constraint is critical. Almost all available bandwidth is used in the Naive and Compensation scenarios. As mentioned above, the number of DR updates increases more significantly in the Compensation scenario with increasing message loss rate. Therefore, we can observe from Figure 3(c) that, compared with the Naive scenario, more DR updates are sent in the Compensation scenario to those receivers with message loss rates $\geq 25\%$; whereas fewer DR updates are sent to receivers with message loss rates $< 25\%$. As a result, we can observe from Figure 4(c) that, compared with the Naive scenario, the TSIs of receivers with high message loss rates are smaller in the Compensation scenario; whereas the TSIs of receivers with low message loss rates are greater. In summary, although similar bandwidth is used in the Naive and Compensation scenarios, the sender in the Compensation scenario sends more DR updates to those receivers with high message loss rates for the purpose of reducing their TSIs to the levels of those receivers with low message loss rates.



(a) Threshold = 5 (b) Threshold = 1 (c) Threshold = 0.2

Fig. 4. Total TSI of receivers in different scenarios



(a) Threshold = 5 (b) Threshold = 1 (c) Threshold = 0.2

Fig. 5. Inconsistency rate of receivers in different scenarios

A receiver can observe the real path of its car and the placed path of the car controlled by the sender. The observed positional relation of these two cars might be different from the real positional relation, which is determined by the real paths of the cars. If the observed and real positional relations are different in a frame, the frame is referred to as an inconsistent frame. The inconsistency rate is defined as the ratio between the number of inconsistent frames and the total number of frames in the simulation. In general, the receivers with higher inconsistency rates observe incorrect positional relations more frequently, and thus is less favorable in terms of the fairness.

Figure 5 shows the inconsistency rates of receivers in different scenarios for different thresholds. As can be seen, the smaller the threshold, the lower the inconsistency rates. Figures 4 and 5, show that the trends of inconsistency rates are highly consistent with those of the corresponding TSIs. Compared to the Original scenario, the inconsistency rates of receivers are reduced significantly in the Naive and Compensation scenarios. In the Compensation scenario, all receivers have almost the same inconsistency rates, which are close to those in the Ideal scenario when the threshold is equal to 5 or 1, but are greater than those in the Ideal scenario when the threshold is equal to *0.2*. However, in the Naive scenario, the inconsistency rates of receivers increase with increasing message loss rates and are always higher than those in the Ideal scenario.

In summary, the budget-based mechanism using the compensation update scheduling algorithm ensures fairness among receivers. With enough bandwidth available, it can promise players the same gaming experience as in the case of no message loss.

VI. Conclusions and Future Work

In this paper, we analytically derive the impact of message loss on TSI of an entity in DVE when using a DR-based update mechanism. To reduce the TSI, the naive and compensation update scheduling algorithms are proposed, taking message loss in the transmission into account. Using these update scheduling algorithms, a budget-based mechanism is developed to send DR updates selectively within network bandwidth constraint. According to our theoretical analysis and experiment results, the budget-based mechanism using the compensation algorithm has advantages over that using the naive algorithm. It can provide fairness among participants in spite of widely varying message loss rates. With enough network bandwidth available, it can promise participants the same experience in the virtual world as in the case without message loss.

In the future, we will extend our work to different movement styles of entities in DVEs and more complex DR extrapolation formulas. Moreover, the message transmission delay and clock synchronization among participants will also be considered.

ACKNOWLEDGMENT

This work was supported in part by Nanyang Technological University under Academic Research Fund Tier 1 Grant RG14/08.

REFERENCES

[1] S. Singhal and M. Zyda, *Networked Virtual Environments: Design and Implementation*. Addison-Wesley, 1999.

[2] D. Miller and J. Thorpe, "Simnet: the advent of simulator networking," *Proc. IEEE*, vol. 83, no. 8, pp. 1114–1123, 1995.

[3] T. Nitta, K. Fujita, and S. Kohno, "An application of distributed virtual environment to foreign language education," in *Procs of the 30th Annual Frontiers in Education - Volume 01*, 2000, pp. 9–15.

[4] T. Hampel, T. Bopp, and R. Hinn, "A peer-to-peer architecture for massive multiplayer online games," in *Procs of 5th ACM SIGCOMM workshop on Network and system support for games (NetGames'06)*, 2006.

[5] IEEE, *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications (2007 revision)*, 2007.

[6] K.-C. Lin, "Dead reckoning and distributed interactive simulation," in *Procs of SPIE Conference (AeroSense'95)*, 1995.

[7] W. Cai, F. B. S. Lee, and L. Chen, "An auto-adaptive dead reckoning algorithm for distributed interactive simulation," in *Procs of the 13th workshop on Parallel and distributed simulation (PADS'99)*, 1999, pp. 82–89.

[8] S. Zhou, W. Cai, B.-S. Lee, and S. J. Turner, "Time-space consistency in large-scale distributed virtual environments," *ACM Trans. Model. Comput. Simul.*, vol. 14, pp. 31–47, January 2004.

[9] S. Aggarwal, H. Banavar, S. Mukherjee, and S. Rangarajan, "Fairness in dead-reckoning based distributed multi-player games," in *Procs of 4th ACM SIGCOMM workshop on Network and system support for games (NetGames'05)*, 2005, pp. 1–10.

[10] D. Hanawa and T. Yonekura, "A proposal of dead reckoning protocol in distributed virtual environment based on the taylor expansion," in *Procs of the 2006 International Conference on Cyberworlds*, 2006, pp. 107–114.

[11] D. Roberts, D. Marshall, R. Aspin, S. McLoone, D. Delaney, and T. Ward, "Exploring the use of local consistency measures as thresholds for dead reckoning update packet generation," in *Procs of the 9th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT '05)*, 2005, pp. 195–202.

[12] S. Aggarwal, H. Banavar, A. Khandelwal, S. Mukherjee, and S. Rangarajan, "Accuracy in dead-reckoning based distributed multi-player games," in *Procs of the 3rd ACM SIGCOMM workshop on Network and system support for games (NetGames'04)*, 2004, pp. 161–165.

[13] L. Pantel and L. C. Wolf, "On the impact of delay on real-time multiplayer games," in *Procs of the 12th international workshop on Network and operating systems support for digital audio and video (NOSSDAV'02)*, 2002, pp. 23–29.

[14] L. Pantel and L. C. Wolf, "On the suitability of dead reckoning schemes for games," in *Procs of the 1st workshop on Network and system support for games (NetGames'02)*, 2002, pp. 79–84.

[15] T. Yasui, Y. Ishibashi, and T. Ikedo, "Influences of network latency and packet loss on consistency in networked racing games," in *Procs of 4th ACM SIGCOMM workshop on Network and system support for games (NetGames'05)*, 2005, pp. 1–8.

[16] A. Malik Khan, S. Chabridon, and A. Beugnard, "A dynamic approach to consistency management for mobile multiplayer games," in *Procs of the 8th international conference on New technologies in distributed systems (NOTERE'08)*, 2008, pp. 42:1–42:6.

[17] G. Cheung and T. Sakamoto, "Construction and scheduling of extrapolated parity packets for dead reckoning in network gaming," in *Procs of the 6th ACM SIGCOMM workshop on Network and system support for games (NetGames'07)*, 2007, pp. 61–66.

[18] C. Faisstnauer, D. Schmalstieg, and W. Purgathofer, "Priority scheduling for networked virtual environments," *IEEE Comput. Graph. Appl.*, vol. 20, pp. 66–75, November 2000.

[19] Y. Yu, Z. Li, L. Shi, Y.-C. Chen, and H. Xu, "Network-aware state update for large scale mobile games," in *Procs of the 16th International Conference on Computer Communications and Networks*, 2007, pp. 563–568.

[20] X. Tang and S. Zhou, "Update scheduling for improving consistency in distributed virtual environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 21, pp. 765–777, June 2010.

[21] Y. Li and W. Cai, "Consistency aware dead reckoning threshold tuning with server assistance in client-server-based dves," in *Procs of 10th International Conference on Computer and Information Technology*, 2010, pp. 2925–2932.

[22] The Open Racing Car Simulator (TORCS). [Online]. Available: http://torcs.sourceforge.net/index.php

APPENDIX

According to the infinite geometric series formula, we can get the following equation, when $0 \leq p < 1$,

$$\sum_{k=1}^{\infty} p^k = \frac{p}{1-p}$$

Consequently, we can get the following equations:

$$\sum_{k=1}^{\infty} k \cdot p^k = p \cdot \sum_{k=1}^{\infty} k \cdot p^{k-1} = p \cdot \frac{\partial(\sum_{k=1}^{\infty} p^k)}{\partial p} = p \cdot \frac{\partial(\frac{p}{1-p})}{\partial p} = \frac{p}{(1-p)^2}$$

$$\sum_{k=1}^{\infty} k^2 \cdot p^k = p \cdot \sum_{k=1}^{\infty} k^2 \cdot p^{k-1} = p \cdot \frac{\partial(\sum_{k=1}^{\infty} k \cdot p^k)}{\partial p} = p \cdot \frac{\partial(\frac{p}{(1-p)^2})}{\partial p} = \frac{(1+p) \cdot p}{(1-p)^3}$$

$$\sum_{k=1}^{\infty} k^3 \cdot p^k = p \cdot \sum_{k=1}^{\infty} k^3 \cdot p^{k-1} = p \cdot \frac{\partial(\sum_{k=1}^{\infty} k^2 \cdot p^k)}{\partial p} = p \cdot \frac{\partial(\frac{(1+p)\cdot p}{(1-p)^3})}{\partial p} = \frac{p \cdot (1+4p+p^2)}{(1-p)^4}$$

Based on the above equations, we can then derive Equations 15 and 16 respectively. These two equations are used in the derivation of Equations 5 and 9 respectively.

$$\sum_{k=1}^{\infty} p^{k-1} \cdot (1-p) \cdot k^3 = \frac{1-p}{p} \cdot \sum_{k=1}^{\infty} k^3 \cdot p^k = \frac{1+4p+p^2}{(1-p)^3} \quad (15)$$

$$\sum_{k=1}^{\infty} p^{k-1} \cdot (1-p) \cdot k^2 = \frac{1-p}{p} \cdot \sum_{k=1}^{\infty} k^2 \cdot p^k = \frac{1+p}{(1-p)^2} \quad (16)$$

Trading Computation Time for Synchronization Time in Spatial Distributed Simulation

Roberto Zunino
Dipartimento di Ingegneria e Scienza dell'Informazione – Università di Trento
The Microsoft Research – University of Trento Centre for Computational and Systems Biology

***Abstract*—We consider a class of models describing generic agents (e.g. macromolecules, small organisms) which are able to travel in space, can sense the surrounding environment, and can react accordingly. In these models, we focus on individual-based simulation. We start with defining a simple centralized simulation algorithm, which we then improve so to develop a distributed algorithm producing the same output. An analytical model is given to estimate the expected speedup of our distributed algorithm depending on several parameters. A main aspect of our approach is that it trades computation time for synchronization time. That is, we allow each node to perform apparently redundant computation whenever this reduces the amount of needed synchronization in such a way that the overall performance improves.**

I. Ideas and Motivation

The main trend in current hardware development suggests that future systems will no longer significantly improve performance because of raw processor speed increase. Rather, major performance improvements will come from making systems more and more parallel. Whether software will be ready to exploit such a high level of paralellism is still uncertain. In this scenario, where idle processors abound, it may be useful to research ways to exploit as many processors as possible to perform a given task, even when the resulting speedup is not optimal. In other words, having abundant processors implies that local computation is *cheap*, and that processors should never be kept idling, unless for power saving reasons.

When parallelizing a task, i.e. performing distributed simulation, one often has to incur into some synchronization overhead because the nodes have to exchange data. Having many idle processors (or nodes) does not however make synchronization costs cheaper; rather, using more nodes requires more synchronization. In this scenario, we would like to make synchronization overhead lower, even if in order to achieve that we need to make (local) computation costs higher. A simple concrete instance of this idea is the following. Let some datum x be available to both nodes n_1 and n_2, which need to obtain the same result $f(x)$. Two possible strategies arise:

- (Communication) One core, say n_1, computes $f(x)$ and then sends the result to the other core n_2.
- (Redundancy) Both cores compute $f(x)$ independently.

Communication is intuitively the *work-efficient* solution: only n_1 performs the actual work, while n_2 is waiting for n_1 to deliver the result. Of course, n_2 can also run something else while it is waiting. On the other hand, redundancy is the *time-efficient* solution, since it completely avoids the synchronization overhead, at the expense of duplicating work.

In a more realistic scenario, such as performing spatial simulation, both nodes will not have to compute the same task $f(x)$. Yet, they might still share intermediate results they need to compute. In that case, a strategy between the above two ones must be chosen, thus choosing between synchronization and computation costs. In this paper we shall design a distributed spatial simulation algorithm by balancing both strategies so to maximize the overall performance. First, we describe our approach in a time-stepped, completely deterministic model (Sect. III). We will then extend our technique to consider probabilistic models (Sect. VI). Finally, we discuss how our results can be adapted to discrete-event simulation (Sect. VII).

II. Models and Centralized Simulation

In order to describe our technique, we start by formalizing the class of models to be simulated. We shall start by focusing on *time-stepped* models, in which time is partitioned in discrete ticks. To model space, instead, we shall use continuous variables. Our aim is then to simulate spatially distributed generic entities which can travel within a given volume S. We shall address these entities as "molecules" to stress our intended application. We shall consider individual-based simulation, that is we want to simulate the individual behaviour of each molecule, rather than the overall behaviour of a large population. Our molecules are stateful, i.e. they are equipped with an internal *state* σ. The overall behaviour of a molecule m is determined by an associated arbitrary algorithm $\mathcal{A}_m$. At each time tick, this algorithm is run with the following inputs:

- the current simulation time t
- the current internal state σ of the molecule
- the current *position* $\vec{x}$ of the molecule
- the current *speed* $\vec{v}$ of the molecule
- the current position and state of the *nearby* molecules — for instance, a protein can react to the presence of nearby enzymes. For our purposes, "nearby" means "within distance δ" for some given constant δ, which does not depend on m.

As output, the algorithm $\mathcal{A}_m$ returns:

- the new internal state of the molecule
- the new speed of the molecule

During simulation, at each time tick we shall update the position of each molecule according to its speed. When doing

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

this, we should also handle the special case of molecules reaching the boundary of the simulation volume. That is, we should e.g. make them stop, or bounce off the boundary. To simplify the discussion, we shall neglect this special case in the rest of the paper.

Below, we present a simple reference centralized simulation algorithm for the class of models discussed above.

Algorithm 1 Reference centralized simulation algorithm

$t \leftarrow 0$
while $t < t_{\mathsf{max}}$ **do**
  for all molecules m **do**
    Let $\mathcal{A}_m$ the algorithm associated to m
    Let $\vec{x}_m, \vec{v}_m, \sigma_m$ be its position, speed, and state
    Let $\vec{x}_1, \sigma_1, \ldots$ be the position and state of the δ-nearby molecules
    $(\sigma'_m, \vec{v}'_m) \leftarrow \mathcal{A}_m(t, \sigma_m, \vec{x}_m, \vec{v}_m, \vec{x}_1, \sigma_1, \ldots)$
    $\vec{x}'_m \leftarrow \vec{x}_m + \vec{v}'_m$
  end for
  for all molecules m **do**
    Update $(\vec{x}_m, \vec{v}_m, \sigma_m)$ with $(\vec{x}'_m, \vec{v}'_m, \sigma'_m)$
  end for
  $t \leftarrow t + 1$
end while

Algorithm 1 is essentially a rather simple time-stepped, individual-based, spatial simulation algorithm. At each main loop iteration, we advance the clock t by one tick, updating molecule position and state according to $\mathcal{A}$. Note that we do not perform an "in-place" update, but instead store the new data in auxiliary variables $\vec{x}'_m, \vec{v}'_m, \sigma'_m$ which are only copied back to $\vec{x}_m, \vec{v}_m, \sigma_m$ after all the molecules have been visited. This is done so to ensure that the result is unaffected by the particular visit order: a molecule m always "sees" other nearby molecules is their previous state, even if they have been already visited.

III. Distributed Simulation – Deterministic Case

In this section, we describe a distributed algorithm to compute *exactly* the same output as our reference Algorithm 1. To keep our presentation simple, we first consider the case of models involving only *deterministic* algorithms $\mathcal{A}_m$. *Probabilistic* algorithms will instead be discussed in Sect. VI.

A. Preliminaries

As a fundamental hypothesis, we assume the following:

Bounded Speed Assumption. At any time, the speed $\vec{v}$ of each molecule is bounded by a constant γ, i.e. $|\vec{v}| < \gamma$.

A nice consequence of this assumption is that, intuitively, information in the model can not be propagated faster than $\lambda = \gamma + \delta$. This fact can be clearly seen through the following thought experiment. Consider two distinct runs of Algorithm 1 on the same model. Since the behavior of molecules is deterministic, we have that the two runs are perfectly *coherent*: at any given time t molecules are in the same state, position, etc. Now, consider instead two parallel runs on two models differing *only* for the state of the single molecule m located at $\vec{x}$. After a tick, molecule m has traveled no more than γ units of space. Also, other nearby molecules (having distance $\leq \delta$ from m) can travel at most γ units of space as well. Let $\lambda = \gamma + \delta$, and consider the portion of space which is at least λ units far from $\vec{x}$, that is $C_1 = \{\vec{y} \mid |\vec{y} - \vec{x}| \geq \lambda\}$. In this subspace, both parallel runs are *coherent*, since no molecule could have observed m in any way. A similar argument supports that after another tick, each molecule at least λ units far from C_1 will be in a coherent state in both parallel runs. By a simple inductive argument, after k ticks the coherent portion of space will include $C_k = \{\vec{y} \mid |\vec{y} - \vec{x}| \geq k \cdot \lambda\}$.

The above bounded speed assumption therefore provides a *lookahead* we can exploit for distributed simulation.

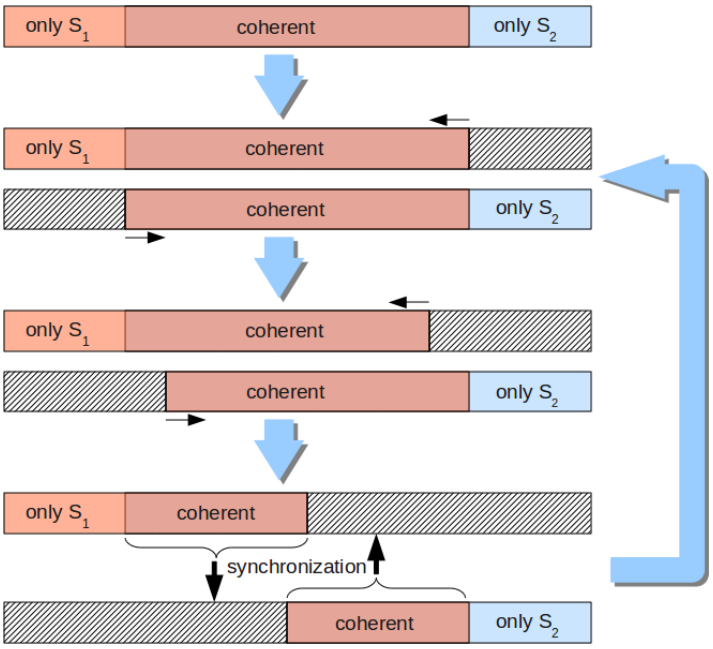


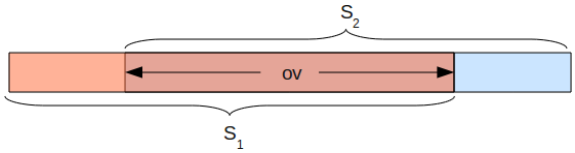
Fig. 1. Scheme of the parallel algorithm (Alg. 2), for two nodes having associated subspaces S_1 and S_2, which overlap in the middle of the volume shown above. Initially, the molecule state in the whole overlapping region is coherent w.r.t. the sequential algorithm (Alg. 1). In each single node, coherency is gradually lost during the Update phase of Alg. 2, yet each molecule can be found in a coherent state in at least one node. This allows us to recover coherency for the whole overlapping region in the Synchronization phase of Alg. 2.

B. The Algorithm

The above reasoning also suggests a way to distribute the simulation among several computational nodes $1 \ldots n$: the pseudo-code is shown in Algorithm 2, which we describe below.

Initialization: First, we *cover* the simulation volume using a family of subspaces $S_1, \ldots, S_n$ where n is the number of the computational nodes. Note that these do not form a partition, in general: we merely require $\bigcup_i S_i = S$. More concretely, one can choose S_i so that they "tile" the simulation

volume. For the sake of simplicity, we shall sometimes refer to the case where S is covered by just two overlapping tiles as shown below.



Update Phase: We run on each node i the centralized Algorithm 1 on the subspace S_i. As the simulation proceeds, intuitively the "inner" part of each S_i will still be coherent with a centralized simulation (over the whole S), while the "outer" part of S_i will slowly lose its coherency at each tick. Indeed, after k ticks the coherent portion of S_i will be (at least) $\mathsf{deflate}(S_i, k \cdot \lambda)$ where

$$\mathsf{deflate}(S_i, \rho) = \{\vec{y} \in S_i | \forall \vec{z} \in S \setminus S_i.\ |\vec{y} - \vec{x}| > \rho\}$$

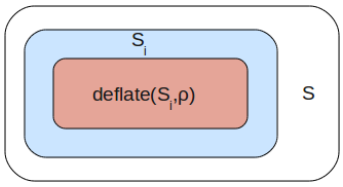


Figure 1 depicts the gradual loss of coherency in the two-tiles case. At every simulation tick, the coherent subspace for each node gets "deflated". Since we want the result of the simulation to be exactly that of our reference centralized simulation algorithm, we must preserve the following invariant: for each point $\vec{x} \in S$, there has to be at least one node i such that $\vec{x}$ is in the coherent portion of S_i at that node. To ensure that invariant, it is enough to guarantee that

$$\bigcup_i \mathsf{deflate}(S_i, t \cdot \lambda) = S$$

Note that the above condition is true when t is less than some threshold value, after which it becomes false. Therefore, the above invariant puts an upper bound k on the amount of time t for which we can run the centralized algorithm.

$$k = \max\ \{t | \bigcup_i \mathsf{deflate}(S_i, t \cdot \lambda) = S\}$$

Note that the time bound k depends on the choice of the family S_i, and the speed bound λ, only.

Synchronization Phase: After k ticks, distributed simulation can not proceed without potentially compromising the exactness of the result. In that case, we perform a synchronization. Each node i communicates with its neighbours, i.e. those nodes j such that $S_i \cap S_j \neq \emptyset$, sending all the molecule data relative to the coherent portion of S_i which is (potentially) not coherent in S_j at node j. More formally, molecule data is sent for the subspace

$$S_i \cap (S_j \setminus \mathsf{deflate}(S_j, k \cdot \lambda))$$

After all the data exchange takes place, each node i has completely restored the coherence of its own subspace S_i. Fig. 1 shows the effect of the synchronization. Therefore, other k ticks can now be simulated in a distributed fashion.

The algorithm then repeats alternating the Update phase and the Synchronization phase until $t = t_{\mathsf{max}}$ is reached. Below we present the pseudo-code for the algorithm discussed so far.

Algorithm 2 Distributed simulation algorithm

```
Pick (S_i)_i such that they cover S
k ← max {k | ⋃_i deflate(S_i, k·λ) = S}
t ← 0
Each node i then runs the following:
while t < t_max do
  {Update phase: run the centralized algorithm on S_i for k·λ ticks}
  t_next ← t + ⌊k·λ⌋
  while t < t_next do
    for all molecules m in S_i do
      Let A the algorithm associated to m
      Let x_m, v_m, σ_m be the position, speed, state of m
      Let x_1, σ_1, ... the position and state of the δ-nearby molecules
      (σ'_m, v'_m) ← A(t, σ_m, x_m, v_m, x_1, σ_1, ...)
      x'_m ← x_m + v'_m
    end for
    for all molecules m in S_i do
      Update (x_m, v_m, σ_m) with (x'_m, v'_m, σ'_m)
    end for
    t ← t + 1
  end while
  {Synchronize phase: exchange data}
  for all neighbour nodes j do
    Send to node j molecule data for zone
        S_i ∩ (S_j \ deflate(S_j, k·λ))
    Receive from node j molecule data for zone
        S_j ∩ (S_i \ deflate(S_i, k·λ))
  end for
end while
```

C. Discussion

Our distributed simulation Algorithm 2 offers the following advantages when compared to the centralized Algorithm 1.

- Each computational node handles a portion S_i of the whole space S. As such, we would expect that simulating a tick in the Update phase has a speedup of roughly $\mathsf{volume}(S)/\mathsf{volume}(S_i)$.
- Each node needs less memory w.r.t. the centralized algorithm. As such, the distributed version should be more cache-friendly than the centralized one.

On the other side, we also have the following drawbacks

- To restore global coherency, nodes have to synchronize every k ticks. This has an impact on the overall performance.
- Since the sets S_i are partially overlapping, some molecules have to be updated on more than a single node. Hence, some work is being duplicated.

Unit name	Description
m	simulation length (meters)
t	simulation time (ticks)
s	wall-clock time (seconds)
mo	number of molecules

Name	Unit	Description
		Parameters of the model
t_{max}	t	total simulation time
λ	m/t	bound on information propagation speed
x, y	m	width, height of the simulation volume
n	mo	molecules in the whole space S
C	$s/(mo \cdot t)$	average molecule state update cost for a single tick
V	$s/\sqrt{mo \cdot t}$	standard deviation for the update
Sy	s/mo	synchronization cost for a single molecule state
ov	m	one-dimensional overlap of S_1, S_2
		Derived quantities
k	t	duration of a single phase
l	1	number of update/sync loops
$\hat{n}$	mo	molecules in a single S_i

Fig. 2. Parameters of the model and derived quantities.

- Whenever most molecules end up in a single subspace S_i instead of spreading over the whole S, we get low performance. The covering $(S_i)_i$ should be indeed picked so to evenly split molecules (rather than volume).

IV. Performance Analysis

In this section we provide an estimate of the performance of our algorithms, in a simple setting. We consider a bi-dimensional space S, of size $x \cdot y$, and provide an approximate analytic model of how much time should our Algorithm 1 and 2 require to produce their results. The parameters used in our model are summarized in Fig. 2, together with other derived quantities. There, we also annotate the units for each quantity.

For the centralized algorithm, we expect the overall time complexity to be proportional both to the total simulation time t_{max} and to the number of molecules n in the whole space S. So, for the centralized algorithm we have

$$\mathsf{CEN} = C \cdot n \cdot t_{\mathsf{max}}$$

where the constant C represents the average time it takes to update the state of a single molecule for a single tick, i.e. the average running time of the algorithm $\mathcal{A}$ which determines the behavior of a molecule.

For the distributed algorithm, we cover the space S with two horizontal tiles, i.e. overlapping in the direction of the x axis for some quantity ov.

$$S_1 = [0, (x + ov)/2] \times [0, y]$$
$$S_2 = [(x - ov)/2, x] \times [0, y]$$

This also determines the number of ticks k used in Algorithm 2 for the update phase. As we have seen, k must be picked so that the coherent portions of all nodes cover the whole simulation volume. Therefore, we have (neglecting truncation):

$$k = ov/(2 \cdot \lambda)$$

The number of update/synchronization loops l required to compute the whole simulation would then be

$$l = t_{\mathsf{max}}/k$$

We estimate the number of molecules present in a single S_i, which we write $\hat{n}$, by normalizing n with respect to $\mathsf{volume}(S_i)$.

$$\hat{n} = n \cdot (x + ov)/(2 \cdot x)$$

The overall cost of the distributed algorithm can then be expressed as the following

$$\mathsf{DIS} = l \cdot (\mathsf{update} + \mathsf{wait} + \mathsf{sync})$$

where update is the cost of a single update phase, sync is the cost of a single synchronization phase. The term wait accounts for the fact that the update phases on both nodes do not take exactly the same amount of time, so the faster node actually has to wait for the slower one. More in detail, the value of update is computed as for CEN, adjusting the number of molecules and the number of ticks:

$$\mathsf{update} = C \cdot \hat{n} \cdot k$$

We approximate the waiting time wait with the standard deviation associated to update. This is proportional to the square root of the molecule updates performed by a node.

$$\mathsf{wait} = V \cdot \sqrt{\hat{n} \cdot k}$$

The synchronization cost sync is instead proportional to the number of molecules in the overlapping volume. We name Sy the constant representing the cost of synchronizing a single molecule between the two nodes.

$$\mathsf{sync} = Sy \cdot n \cdot ov/(2 \cdot x)$$

We can now discuss the speedup of our distributed algorithm,

$$\mathsf{speedup} = \mathsf{CEN}/\mathsf{DIS}$$

From the formulas above, after some simplification one can observe that the speedup does not depend from the parameters t_{max} and y. This was expected, since both CEN and DIS are proportional to t_{max}, while y is immaterial since the subspaces S_1, S_2 overlap on the x axis. Accordingly, in the following discussion we shall neglect the parameters t_{max} and y.

A. Evaluation

We now discuss how the expected speedup is affected by the parameters of the model. More in detail, we want to study how relevant is the choice of the overlap ov of the subspaces S_1, S_2, in a concrete biology-inspired example.

We consider 10^4 molecules (n), each one of diameter $\sim 100 \cdot \mathring{A} = 10^{-8} m$, in a two-dimensional simulation volume S which is x by y wide. As anticipated, the actual value

of y is irrelevant. Instead, we let $x = 5 \cdot 10^{-4} \cdot m$ so to obtain an average linear molecule density along the x axis of $1/5$. As above, S is covered by two "tiles" along the x axis. We further assume that information can not propagate faster than one molecular diameter per tick, hence we let $\lambda = 10^{-8} \cdot m/t$. As for the parameters C, V, Sy, we pick them so that synchronization is much expensive than computation, as it happens in practice. We let $C = 10^{-7} \cdot s/(mo \cdot t)$, $V = 10^{-8} \cdot s/\sqrt{mo \cdot t}$, and $S = 10^{-3} \cdot s/mo$.

We now plot the expected speedup depending on the overlap ov between S_1, S_2. In order to make it easy to relate the results to the execution of Algorithm 2, in our plot we choose k (i.e. how many ticks are simulated within each Update phase) instead of ov as the abscissa. Recall that $k = ov/(2 \cdot \lambda)$, so this choice just amounts to using a different scale for the abscissa.

The resulting expected speedup is shown in Fig. 3. The upper plot shows that the performance quickly reaches its maximum for $k \simeq 7$, i.e. when S_1, S_2 have a small intersection. In this case the distributed algorithm is about $\%43$ faster than the centralized one. The lower plot instead shows what happens when k is three magnitude orders larger, i.e. when subspaces S_1, S_2 largely overlap. The speedup slowly decreases, reaching 1 for $k \simeq 15 \cdot 10^3$. Hence we have $\mathsf{DIS} = \mathsf{CEN}$ when the subspaces S_1 and S_2 are 60% overlapping.

We also want to observe how the expected speedup is sensitive to the actual value of the parameter Sy, i.e. to the synchronization overhead. To this purpose, we reuse the same parameters defined above, but for Sy which is now varying. For each Sy value, we define k (or equivalently, ov) so that the speedup is maximized. The resulting plot is in Fig. 4. For very low values of Sy, i.e. when synchronization is very cheap, the distributed algorithm reaches near-optimum speedup: indeed, in that case one can synchronize the nodes at each tick. For large values of Sy, instead the synchronization overhead completely negates the advantages of the distributed algorithm.

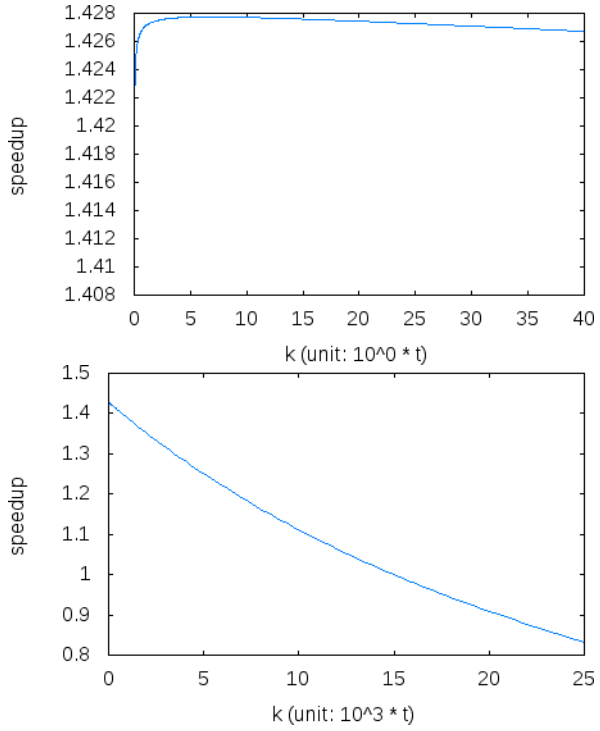


Fig. 3. Expected speedup as a function of k, plotted at different scales. Other parameters: $n = 10^4 \cdot mo$, $C = 10^{-7} \cdot s/(mo \cdot t)$, $V = 10^{-8} \cdot s/\sqrt{mo \cdot t}$, $Sy = 10^{-3} \cdot s/mo$, $x = 5 \cdot 10^{-4} \cdot m$, $\lambda = 10^{-8} \cdot m/t$. The maximum speedup is $\simeq 1.43$, reached when $k \simeq 7$.

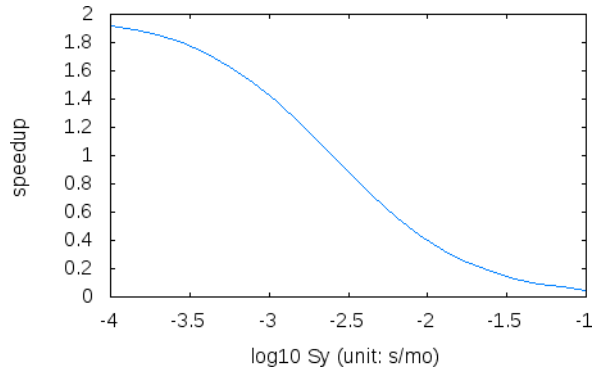


Fig. 4. Expected speedup as a function of Sy. Other parameters are as in Fig. 3, except for k which is choosen so to maximize speedup.

V. Some Extensions to the Model

So far, we considered a simple model in which molecules can essentially perform one operation, namely to change their internal state according the presence of nearby molecules. Several more complex operations can be added to the model without compromising the correctness of the distributed algorithm 2. As long as the bound λ can still be determined, providing the required lookahead for the distributed algorithm, we foresee no large difficulty in extending the model with:

- creation (spawning) of new molecules
- degradation (destruction) of existing molecules
- signaling (communication) between existing molecules
- chemical reactions (e.g. $A + B \rightarrow C + D$)

Chemical reactions do require a little care to be handled, in that the same molecule can not act as a reactant (i.e. be consumed) for two distinct reactions. This issue already shows up in the centralized setting: the simulator has to somehow resolve the conflict, possibly also examining the state of molecules. For instance, in the real world, conflicts of course do not occur since reactions require close proximity to specific sites and molecules being physical bodies can not compenetrate. Restricting the movement of molecules in our model accordingly would avoid conflicts between reactions. Another possibility would be to resolve conflicts using priorities, defining a total ordering between (instances of) reactions. In that case, the simulator can pick the highest-priority reaction.

Whether the conflicts are avoided or resolved, as long as it is possible to decide whether a (instance of a) reaction can be fired by examining to a λ-neighbourhood of our simulation volume, then we do obtain the required lookahead we need to apply our distributed algorithm.

The next two sections extend our technique to further kinds of models.

VI. Distributed Simulation – Probabilistic Case

So far, we considered deterministic algorithms $\mathcal{A}_m$, only. This requirement is essential in ensuring that every node is perfectly coherent w.r.t. the centralized simulation. Indeed, if we allow $\mathcal{A}_m$ to toss a coin, this needs to be duplicated on all nodes which hold a copy of m. If we simply call a random number generator for realizing this, in each node we will likely end up with a different result, compromising coherency.

In short, we need to distribute the same random number generator, so it generates the same results for the same molecules in distinct nodes. For this, we propose the following.

- Before Algorithm 2 is started, a "master" node generates a random bit string seed, and broadcast to each other node. Also, a unique numeric ID_m is associated to each molecule m.
- Whenever algorithm $\mathcal{A}_m$ invokes the random number generator for the i-th time at time tick t, it receives

$$\mathsf{hash}(\mathsf{seed}, ID_m, t, i)$$

where hash is a "strong" hashing function.

The intuition is that the sequence $\mathsf{hash}(\mathsf{seed}, ID_m, t, 1)$, $\mathsf{hash}(\mathsf{seed}, ID_m, t, 2), \ldots$ is, from the point of view of $\mathcal{A}$, indistinguishable from a pseudo-random sequence since seed is unknown to $\mathcal{A}$. Also, this sequence will be coherently generated by all nodes. Hence, Algorithm 2 can still be applied. This is because, essentially, once the seed has been fixed the rest of the algorithm is completely deterministic. So, the above reasoning actually reduces probabilistic models to deterministic ones plus an initial seed generation and distribution.

The choice of the hash function is important to ensure a "good" level of pseudo-randomness. One might want to employ for this purpose a cryptographic-strong hash [1], such as SHA1. We foresee that for simulation purposes, one might also want to trade some of the "strength" of the hash for achieving better performances.

Other techniques can also be used to perform distributed pseudo-random number generation [2], [3]. For our aims, it suffices to assign each molecule an independent sequence of random bits.

VII. Towards Discrete-Event Simulation

In this section we discuss how we can extend our main technique from time-stepped models to the more general case of discrete-event simulation (DEVS) [4]. This would enable us to distribute the simulation of models where, for instance, the interactions between molecules are driven by stochastic laws, beyond some constraints on their positioning. We now discuss some ideas about how to adapt Algorithms 1 and 2 to DEVS, as well as some issues that arise when doing this.

First, we have to adapt the centralized reference algorithm. Recall each molecule m has an associated algorithm $\mathcal{A}$ defining its behaviour. For DEVS, we make $\mathcal{A}$ output the time for the next event related to m in addition to the other outputs shown in Algorithm 1. Without loss of generality, we can assume $\mathcal{A}$ to be deterministic, as discussed in Sect. VI. Then, we adapt Algorithm 1 as follows.

- Compute the time-to-next-event for each molecule. Store these events in a priority queue.
- At each loop iteration, extract from the queue the first event.
- Advance the simulation clock to the time of the first event, moving the molecules accordingly.
- Perform the event. This involves calling algorithm $\mathcal{A}$ for the affected molecule(s), which may update their internal state. (Note that algorithm $\mathcal{A}$ at this point may also decide that the new positioning is no longer compatible with the intended event, and therefore no action is to be taken.)
- Running $\mathcal{A}$ generates new time-to-next-events, so we update the event queue accordingly.
- Continue with the next loop iteration.

We now adapt our distributed simulation Algorithm 2 to DEVS. We still rely on having a strict bound λ on the information flow speed in the simulation volume, as we did for the time-stepped case. We then proceed with the *update* and *synchronization* phases as follows. In the *update* phase, we make each node run the centralized algorithm discussed above on a subspace S_i, where $(S_i)_i$ cover the whole simulation volume S. Again, as in the time-stepped case, we keep on running the centralized algorithm as long as possible, provided the *coherent* regions (w.r.t. the centralized algorithm) of each subspace still cover S. For DEVS, this means that we stop whenever advancing the node-local clock to the next event in that node would violate the covering condition. More concretely, nodes stop when their next event has time greater than some threshold t_{next}, which only depends on λ and $(S_i)_i$. After all the nodes stop, we can enter the *synchronization* phase. Nodes now exchange molecule data so that each node restores coherency over its own whole subspace. The event queue is updated according to the updated molecule state. After this is done, we proceed with the next *update* phase.

The above discussion however hides a couple of important technical issues, which were not a concern in the time-stepped case, but become relevant for DEVS. First, note that when the synchronization phase is started, the node-local clocks are at *different* times, in general. Indeed, node number i will stop at time t_i when the next scheduled event *in its own subspace* S_i is after t_{next}. Therefore, before synchronization can be performed, each clock has to be advanced to the same value $\max\{t_i \mid i \in I\}$ where t_i is the local clock of node i, so moving molecules accordingly. After that, exchanging actual molecule data is straightforward. Note that this procedure does not advance clocks beyond the threshold t_{next}, so the coherent regions of each subspace S_i still cover the whole S. Therefore, synchronization indeed restores coherency to each node.

A second more subtle technical issue arises from the presence of *rounding errors*. To understand it, let us consider the the time-stepped case first. There, both the centralized and

distributed simulation algorithms update the position x of a molecule performing the assignment $\vec{x}\,' \leftarrow \vec{x}+\vec{v}$ where $\vec{v}$ is the molecule speed. In a concrete implementation, floating point numbers are used to represent these vectors, hence introducing rounding errors. However, both algorithms will sum exactly the *same* values for $\vec{x}$ and $\vec{v}$, so the resulting $\vec{x}'$ will be identical — with the same rounding error. Hence, both algorithms at the same simulation time will be perfectly coherent in their data, bit-by-bit. Now, let us discuss the DEVS case. Here, position update is done by the assignment

$$\vec{x}\,' \leftarrow \vec{x} + \vec{v} \cdot (t_{\mathsf{n}} - t)$$

where t is the current time, and t_{n} is the time of the next event. Suppose now that events in the centralized algorithm occur at times t_1, t_2, t_3. When distributing the simulation over nodes, it is possible for one node to observe only the events at t_1, t_3 since the event at t_2 did not occur in its own subspace. Below, we write the position updates for a generic molecule which are performed by the centralized and distributed algorithms (assuming constant $\vec{v}$):

$$\begin{array}{ll} \vec{x}\,'_{\mathsf{centralized}} & \leftarrow \big(\vec{x} + \vec{v} \cdot (t_2 - t_1)\big) + \vec{v} \cdot (t_3 - t_2) \\ \vec{x}\,'_{\mathsf{distributed}} & \leftarrow \vec{x} + \vec{v} \cdot (t_3 - t_1) \end{array}$$

Floating point arithmetic offers *no* guarantee that the above values will be perfectly identical — rather, it merely guarantees a small relative error w.r.t. the exact result, implying the two positions above differ by a small relative amount. This however means that centralized and distributed simulation algorithms will not be coherent "at the bit level", but only in a more approximated way.

It is arguable whether rounding errors can have any relevant effect on the outcome of the simulation. From one hand, the initial state of the model we are simulating often has an associated error which is larger than the one introduced by rounding. So, introducing a small perturbation could seem harmless. On the other hand, losing the *exactness* of our simulation is somehow disturbing: the result is now affected by performance-tuning parameters such as the choice of the $(S_i)_i$ covering or the number of nodes we are using. This also undermines the reproducibility of the simulation: to achieve the same results we must be careful in choosing the same $(S_i)_i$, etc. Unfortunately, the worst-case consequence is rather unpleasing: in a chaotic system even a very small discrepancy can quickly be amplified and lead to a drastically different outcome. Also, a molecule might even "get lost" or duplicated because of rounding if it happens to be near the boundary of the region which is being synchronized between two nodes.

In order to address this issue, it may be convenient to define the centralized simulation in a slightly different way. Recall the formula used above for updating the position of molecules (t = current time, t_{n} = time of the next event in queue):

$$\vec{x}\,' \leftarrow \vec{x} + \vec{v} \cdot (t_{\mathsf{n}} - t)$$

We now redefine the above position update in alternative way. We keep track, for each molecule m, of its current position $\vec{x}$ (as before), as well as the time t_{l} of the *last event affecting* m and its *position at that time* $\vec{y}$. Since the speed $\vec{v}$ of molecule m is unaffected by events unrelated to m, we can assume $\vec{v}$ to be constant between t_{l} and t_{n}. So we can write:

$$\begin{array}{ll} \vec{x}\,' \leftarrow \vec{y} + \vec{v} \cdot (t_{\mathsf{n}} - t_{\mathsf{l}}) & \\ \vec{y}\,' \leftarrow \vec{y} & \text{if the next event is relevant to } m \\ t_{\mathsf{l}} \leftarrow t_{\mathsf{n}} & \text{if the next event is relevant to } m \end{array}$$

The above does not reuse the old position $\vec{x}$ at time t, but instead recomputes the position of the molecule m starting from the time of the last event affecting it (t_{l}). When considering exact numeric operations ("infinite precision") the above assignment for $\vec{x}\,'$ is equivalent to the one we met previously. However, when considering floating point operations, the one above guarantees that events unrelated to m do not affect the rounding error affecting $\vec{x}\,'$. Therefore, this makes it possible to restore bit-level equivalence between centralized and distributed simulation.

VIII. Conclusions and Related Work

We believe that there is still room for improvement in Algorithm 2. For instance, during the Update phase, the whole subspace of the node at hand is updated, including the portion of it which has lost coherency with respect to the centralized simulation. It could be beneficial to keep track of this and avoid updating the incoherent portion, so that update phases become slightly cheaper at each iteration, until a synchronization is in order.

Further, there is no actual constraint to reuse the same covering of the simulation volume after each synchronization. That is, after global coherency is restored, nodes could benefit to agree on a new covering. For instance, if many molecules moved to the same subspace, making the overall molecule density significantly non-uniform, it would be better if that dense subspace is split in more subspaces, so to balance work load. Similarly, low-density subspaces could be merged together. Ideally, each node would then be assigned a *new* subspace with roughly the same number of molecules, with the aim to keep the wait component of the cost as small as possible. This would require to use some adaptive subspace covering heuristics.

The approach to distributed simulation studied in this paper falls in the SRIP class (Single Replications In Parallel) [5] in that we focus on exploiting parallelism to reduce the time required to produce a single simulation trace. In many applications, such as simulating stochastic models derived from systems biology, it is useful to produce many traces at the same time and run statistic analysis on them. In that case, the task is embarrassingly parallel, and MRIP simulation (Multiple Replications In Parallel) can be applied with nearly-optimum speedup. However, having a fast method of producing single traces is still rather valuable. Indeed, defining a complex model to be simulated is unfortunately an error-prone task, so one often has to "debug" the model by comparing some output traces to some specific expected behaviour (e.g. "that enzyme should not be so abundant"). At this stage, we want to be

able to generate a few traces quickly, so that we can detect modeling errors early.

Distributed algorithms for discrete-event simulation can be divided in *conservative* and *optimistic* ones according to, roughly speaking, whether they always ensure coherency with a centralized algorithm by using synchronizations (e.g. locks) so no node clock runs too far ahead the others, or instead they could temporarily lose coherency and let clocks run ahead but they are prepared to rollback to a previous time in which coherency held. In this respect, our algorithm is a *conservative* one in that we never need to perform rollbacks, and we do synchronize our clocks. In an optimistic algorithm, work is "wasted" whenever a rollback is needed. In our algorithm, in a sense, we "waste" work by duplicating computation in several nodes.

The spatial model we consider is tailored for reaction-diffusion *individual-based* simulation. Instead, for reaction-diffusion *population-based* simulation a commonly employed algorithm is the Next Subvolume Method [6]. In the NSM model, the space is partitioned in subvolumes, and for each subvolume only the population count for each species is kept. In equivalent terms, the position of each molecule is not stored exactly, but instead it is approximated with its enclosing subvolume. So, two molecules falling in the same subvolume are indistinguishable, and the population count completely describes the state of the simulation. Stochastic rates are used to drive intra-subvolume reactions and inter-subvolume diffusions, using an adaptation of the Next Reaction Method [7]. This model offers many possibilities for parallelization [8].

The Gillespie Multi-Particle method (GMP) [9] also uses subvolumes for modelling reaction-diffusion stochastic simulation, using a variant of the Gillespie direct method [10]. Like the NSM, it performs population-based stochastic simulation, except that it uses separate diffusion and reaction phases. In the diffusion phase, all subvolumes perform diffusion. By comparison, in the NSM diffusion is performed by two subvolumes at each distinct event. GMP therefore separates diffusion events, which happen at regular intervals, from reaction events which are stochastic.

Many other spatial models and related simulation methods exist [11], [12], working at distinct scales, and modelling diffusion in different ways (e.g. ODEs, Browninan motion). In this respect, our models make use of a continuous space where movement is determined by an arbitrary algorithm. While this design choice provides a large amount of flexibility, one might wish to restrict the attention to more specific cases when this improves performances. Also, our molecules are modelled as points, making them occupy no volume. To study molecular crowding properties [11] we should extend our technique to impenetrable molecules as hinted in Sect. V. Some ideas from the Shape calculus [13] might be borrowed for this purpose.

We believe that it is possible to adapt our techniques to high-level modelling languages such as Space π [14]. Intuitively, we could borrow some machinery from a parallel abstract machine [15] for β-binders [16], and use the spatial information to derive the bound λ which provides the lookahead. We leave this as future work.

REFERENCES

[1] B. Preneel, "Analysis and design of cryptographic hash functions," Ph.D. dissertation, Katholieke Universiteit Leuven, 1993.

[2] C. Tan, "On parallel pseudo-random number generation," in *Computational Science ICCS 2001*, ser. Lecture Notes in Computer Science, V. Alexandrov, J. Dongarra, B. Juliano, R. Renner, and C. Tan, Eds. Springer Berlin / Heidelberg, 2001, vol. 2073, pp. 589–596.

[3] A. Srinivasan, M. Mascagni, and D. Ceperley, "Testing parallel random number generators," *Parallel Computing*, vol. 29, no. 1, pp. 69 – 94, 2003.

[4] J. Banks, J. Carson, B. L. Nelson, and D. Nicol, *Discrete-Event System Simulation, Fourth Edition*. Prentice Hall, 2004.

[5] K. Pawlikowski, V. Yau, and D. McNickle, "Distributed stochastic discrete-event simulation in parallel times streams," in *Proc. Winter Simulation Conference (WSC)*, 1994.

[6] J. Elf and M. Ehrenberg, "Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases," *Systems Biology, IEE Proceedings*, vol. 1, no. 2, pp. 230 – 236, Dec. 2004.

[7] M. A. Gibson and J. Bruck, "Efficient exact stochastic simulation of chemical systems with many species and many channels," *The Journal of Physical Chemistry A*, vol. 104, no. 9, pp. 1876–1889, 2000.

[8] M. Jeschke, A. Park, R. Ewald, R. Fujimoto, and A. M. Uhrmacher, "Parallel and distributed spatial simulation of chemical reactions," in *Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation*, ser. PADS '08. IEEE Computer Society, 2008, pp. 51–59.

[9] J. V. Rodriguez, J. Kaandorp, M. Dobrzynski, and J. Blom, "Spatial stochastic modelling of the phosphoenolpyruvate-dependent phosphotransferase (PTS) pathway in escherichia coli," *Bioinformatics*, vol. 22, no. 15, 2006.

[10] D. T. Gillespie, "Exact stochastic simulation of coupled chemical reactions," *The Journal of Physical Chemistry*, vol. 81, no. 25, pp. 2340–2361, 1977.

[11] K. Takahashi, S. Nanda, V. Arjunan, and M. Tomita, "Space in systems biology of signaling pathways: towards intracellular molecular crowding in silico," *FEBS letters*, vol. 579, no. 8, pp. 1783–1788, 2005.

[12] A. T. Bittig and A. M. Uhrmacher, "Spatial modeling in cell biology at multiple levels," in *Winter Simulation Conference (WSC), Proceedings of the 2010*, dec 2010, pp. 608 –619.

[13] E. Bartocci, F. Corradini, M. R. Di Berardini, E. Merelli, and L. Tesei, "Shape Calculus. A Spatial Mobile Calculus for 3D Shapes," *Scientific Annals of Computer Science*, vol. 20, pp. 1–31, 2010.

[14] M. John, R. Ewald, and A. M. Uhrmacher, "A spatial extension to the π calculus," *Electronic Notes in Theoretical Computer Science*, vol. 194, no. 3, pp. 133 – 148, 2008, proceedings of the First Workshop From Biology To Concurrency and back (FBTC 2007).

[15] S. Leye, A. Uhrmacher, and C. Priami, "A bounded-optimistic, parallel beta-binders simulator," in *Distributed Simulation and Real-Time Applications, 2008. DS-RT 2008. 12th IEEE/ACM International Symposium on*, oct 2008, pp. 139 –148.

[16] P. Degano, D. Prandi, C. Priami, and P. Quaglia, "Beta-binders for biological quantitative experiments," *Electronic Notes in Theoretical Computer Science*, vol. 164, no. 3, pp. 101 – 117, 2006, proceedings of the 4th International Workshop on Quantitative Aspects of Programming Languages (QAPL 2006).

Framework for Simulation of Hybrid Systems: Interoperation of Discrete Event and Continuous Simulators Using HLA/RTI

Changho Sung* and Tag Gon Kim†
Department of Electrical Engineering
KAIST, Daejeon, Republic of Korea
Email: chsung@smslab.kaist.ac.kr*, tkim@ee.kaist.ac.kr†

***Abstract*—A hybrid system is a combination of discrete event and continuous systems that act together to perform a function not possible with any one of the individual system types alone. A simulation model for the system consists of two sub-models, a continuous system model, and a discrete event model, and interfaces between them. Naturally, the modeling/simulation tool/environment of each sub-model may be different and specific to the model types and the associated modeling formalisms. This paper proposes a framework for simulating hybrid systems by means of interoperation between existing simulators for a continuous model and a discrete event model using a High Level Architecture (HLA). Each simulator is executed using different simulation algorithm, and pre-simulation methodology is applied for synchronization of the different simulators. The main advantage of the proposed approach is the reuse of simulation models, which are developed in their own simulation environments of heterogeneous types. The proposed framework was applied to water level simulation.**

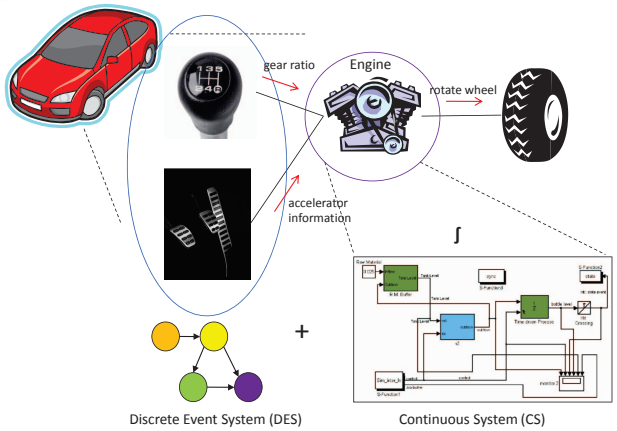


Fig. 1. Hybrid system

I. Introduction

Systems modeling and simulation process vary according to the types and modeling objectives of the systems. Given modeling objectives, modelers should determine modeling levels and simulation tools. For example, continuous models relate to the change of values via time, and discrete event models are concerned with state transitions by events. On the other hand, a hybrid system is defined as a mixture of continuous and discrete time components that act together to perform a function that is impossible with any one of the individual system types alone. An automobile is a kind of hybrid system, and each component is shown in Fig. 1. The automobile is a combination of several sub-systems. Among them, the engine system is represented by several differential equations. The equations describe compressible flow through the engine, heat transfer, mechanical vibration, overall engine dynamics, etc. On the other hand, a gear system is represented by discrete event models. The gear ratio changes from 1 to 5 or 6, and the change occurs manually or automatically. The information becomes a parameter for solving the differential equations of the automobile engine. In order to model and simulate a hybrid system, such as an automobile, each sub-systems is modeled in its own modeling environment, and the combined simulation method is needed.

Simulating models developed in different methodologies and environments is difficult because their simulation algorithms and information are disparate from each other. To overcome the difficulty in simulating hybrid systems, modeling methods based on a combined formalism, which has both discrete event and continuous features, are proposed. However, the combined modeling method is limited by the non-reusability of existing simulators. In addition, efforts have been made to integrate the simulators with a specific interface, but the method also needs modification of the existing simulator for the interface. To solve the limitations, we propose an interoperation framework for the simulation of hybrid systems with existing discrete event and continuous simulators.

In general, the modeling of hybrid systems may require distributed simulation due to system complexity, reusability, and interoperability requirements. In order to simulate different types of models, developed simulators should interoperate with others developed from various environments. Interoperation could be achieved by using the High Level Architecture (HLA) developed by the US Department of Defense (DoD) for modeling and simulation [1] [2] [3]. The Run-Time Infrastructure (RTI) is an implementation of the HLA interface, and it supports developers who use HLA services [4]. This paper proposes an interoperation framework for the simulation of hybrid systems using HLA/RTI. The framework includes hybrid simulation algorithms and considers existing simulation models.

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

This paper is organized as follows. Section II presents several related works and previous approaches to the simulation of hybrid systems. Section III explains the simulation method for hybrid systems, and the proposed framework for hybrid simulation is described in Section IV. Section V demonstrates an example of a water tank control system and its simulation results. Finally, Section VI concludes the discussion.

II. Related Work

In the past years, some effort has been made to simulate hybrid systems. Most of the research has concentrated on specifying a unified methodology for hybrid simulation. We call the methodology *stand-alone* simulation. The *stand-alone* simulation of hybrid systems means that the systems are modeled and simulated by only one modeling tool or methodology without the help of any others. For example, the following tools are used for stand-alone simulation and modeling systems: MATLAB/Simulink, Ptolemy II [5], AnyLogic [6], and PowerDEVS [7]. The *SimEvents* toolbox [8] supports MATTLAB/Simulink supports discrete event simulation, and the *Event Translation* block enables communication between *SimEvents* and other continuous operational blocks. Ptolemy II is a computational framework for embedded systems that focuses on concurrent systems, and HyVisual [5] is a hybrid system modeling tool that provides a visual syntax. AnyLogic is a tool for modeling and simulating hybrid systems and a way of HLA support integration in the tool simulation engine. PowerDEVS proposes discrete event methods for the numerical integration of ordinary differential equations (ODEs). These approaches commonly propose the use of a unique language for the specifications of the overall system, which consists of different model types. These require new modeling formalisms or tools. Therefore, existing models may not be reusable, and the model development process has been long.

In addition, some research has simulated models separately, each of which is communicated by procedure calls. We call the methodology *integrated* simulation. The methodology is expressed by MATLAB/Simulink interfaces, such as CODIS [9] and BCVTB [10], and is applied to most of the hybrid control systems. CODIS is a framework for simulating continuous and discrete systems, and it has a co-simulation bus to integrate a SystemC model and a MATLAB/Simulink model. BCVTB is based on the Ptolemy II software environment, and it integrates with a MATLAB/Simulink model. These methodologies use a specific interface between two models, such as Inter-Process Communication (IPC) and function calls. Two models are executed using such an interface in one process and tightly coupled. Thus, certain limitations have existed on the simulation of models developed in different modeling environments.

Another simulation method consists of the interoperation between discrete event system models and continuous system models using HLA/RTI. The models are spatially separated explicitly. In a few studies, the HDEVSimHLA [11] framework provides an HLA/RTI interface for interoperating the DEVS models and the MATLAB/Simulink models. The framework uses the time management service of HLA/RTI for time synchronization between heterogeneous models, and it uses the analog-event and event-analog functions for data exchange. In addition, AnyLogic [6] and MATLAB provides HLA/RTI interfaces as a package of the application. Developers have to use the HLA support modules and implement new interface codes in each model. This interoperation methodology is the most efficient method for reusing existing models and modeling systems in different environments. In these researches, however, existing models may not also be reusable, because the models are modified to use HLA/RTI, and new interface codes have to be inserted.

In comparison with previous research, the contribution of this paper consists in the concept of reusing existing models without any modifications. The problem common to previous studies has been the difficulty of reusing existing continuous and discrete event models, which are developed in different tools, and inserting specific interfaces into them. To overcome the problems, this paper proposes using a distributed hybrid simulation framework for the interoperation of existing models developed in different tools. The framework supports interoperation of such models via the standard interface of HLA. It also supports time synchronization between models and data conversion for communicating with each other. A main purpose of the proposed approach is to exploit the reusability of existing continuous and discrete event models, each of which is simulated in independent simulation environment/tools.

III. Hybrid Simulation Algorithm

In general, a discrete event model is executed by an event-driven approach. It advances time asynchronously—that is, it skips time when there is no change in the state of the system and examines the changes. Therefore, it is important to find the minimum next-event time of the discrete event model, because interval time between current and next-event time is not considered. It takes the execution time of an event from the top of a sorted stack, calculates its effect on the system state, schedules dependent events by placing them into appropriate places in the event stack, and advances the simulation clock to the next scheduled event. On the other hand, a continuous model is commonly executed by the discrete-time simulation method. It advances synchronously—that is, it uses the continuous state at some time, t, to compute the value of a continuous state variable at some short time later at $t + \Delta t$. The continuous variable solver then advances the simulation clock to $t + \Delta t$ and continues [12]. The time step may be determined by the numerical analysis method of the continuous model.

Simulation of a hybrid system goes beyond the simple numerical integration of the continuous variables and specification of the particular interactions that may occur among continuous and discrete state variables; there is a fundamental mismatch between the way time is advanced by a discrete event-driven versus a continuous variable integrator. In order to solve the mismatch, simulation time synchronization between a discrete event and a continuous model is required. However,

the next time of the continuous model for scheduling is not predictable, unlike the time of discrete event model. To solve the problem, we use a pre-simulation method for scheduling the continuous model.

A. Pre-Simulation

Simulation time synchronization requires a chronological order of logical times between continuous and discrete event simulations. To synchronize the simulation times with different time advance mechanisms we introduce the concept of pre-simulation for continuous simulation. Pre-simulation [11] is continuous simulation except that it occurs within a predefined time window to unify the time advance mechanism between two simulations, and the next scheduled time is located through pre-simulation. Hence, *pre-simulation* is defined as *simulating a continuous time model in advance to find the next scheduled time of the model*. To explain this simulation method, we used the concept of *state event* and *time event* [13]. A state event is triggered by the system status, the fulfilling certain conditions (so-called state condition)—e.g., the crossing of a state variable through a prescribed threshold. A time event is a scheduled event whose realization time is known in advance. A time event occurs at a given point in time, independent of the continuous-time state of the model. Therefore, a time event is predictable.

Algorithm 1 shows the pre-simulation of a continuous model. Let t_C denotes the current simulation time and t_R be the next request time. The continuous model is simulated between t_C and t_R, and the interval is a predefined time window. If a state event is detected within the time window, the state event occurs, and the event time, t_E, is located in the threshold point. t_E is set to the next scheduled time. On the other hand, if not, a time event occurs and the next scheduled time is set to t_R. After simulation, reinitialization of state variables is needed. The pre-simulation process allows us to control the time advancement of a continuous model.

Algorithm 1 Pre-Simulation of Continuous Model

procedure PRE-SIMULATION(t_C, t_R)

1: t_C : current simulation time
2: t_R : next requested time
3: t_E : state event time
4: t_N : next scheduled time
5: simulation in $[t_C, t_R]$
6: **if** state event detected **then**
7: send state event and the time(t_E) is located
8: $t_N \leftarrow t_E$
9: **else**
10: send time event
11: $t_N \leftarrow t_R$
12: **end if**
13: reinitialization of state variables
14: **return** t_N

B. Time Advancement of Continuous Model

Let ΔT denotes the time window to find the next scheduled time, and let s be the state of a continuous model. The state, s, is either *ACTIVE* or *PASSIVE*. An *ACTIVE* state means that the continuous model is in the execution state for generating a state event or time event, and the *PASSIVE* state is the waiting state for events from the discrete event model. Algorithm 2 shows the time advance mechanism of a continuous model. The next-event time of a continuous model is the result of pre-simulation operation in an *ACTIVE* state. In the *PASSIVE* state, the pre-simulation process is not needed, and the next time is infinity because the continuous model does not advance its time until it receives data from the discrete event model.

Algorithm 2 Time Advancement of Continuous Model

procedure TIME-ADVANCE

1: t_C : current simulation time
2: t_N : next scheduled time
3: s : state of continuous model {ACTIVE, PASSIVE}
4: ΔT : time window
5: initialization
6: $s \leftarrow$ ACTIVE
7: **loop**
8: **if** s is ACTIVE **then**
9: $t_R \leftarrow t_C + \Delta T$
10: $t_N \leftarrow$ Pre-Simulation(t_C, t_R)
11: **else**
12: $t_N \leftarrow \infty$
13: **end if**
14: request next time advance(t_N)
15: **end loop**

C. Hybrid Simulation

1) Synchronization: Using the pre-simulation method, we can find the next scheduled time of a continuous model. The time scheduler calculates the next time of the simulators and sends the time information and interaction message between the models. In hybrid simulation, there are two types of interaction, as shown in Fig. 2.

- A discrete event changes the relation of a continuous variable.
- A continuous variable reaches a threshold point, and it causes a discrete event.

We can predict the next-event time of a discrete event model, and the time, t_1, is requested to time scheduler. This means that the discrete event model has no events and no changes of state until t_1. A continuous model executes a pre-simulation with time window ΔT until t_1 to check state events. If there is no state event in the continuous model, the discrete event on t_1 is transferred to the continuous model, and the relation of the continuous variable, S_cont, is changed. If a state event occurs during pre-simulation, the continuous model causes a discrete event at the threshold point, t_2. The event

changes the state of the discrete event model, and the model is re-scheduled.

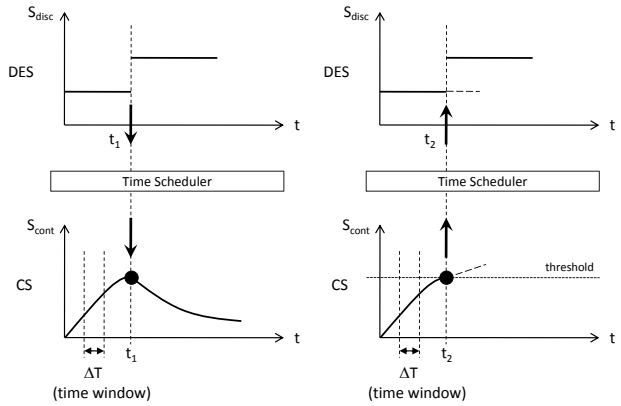


Fig. 2. Pre-simulation of CS and time scheduling

2) Data Conversion: A discrete event model requires a discrete input event for its simulation, and it generates a discrete output event. A continuous model needs an input analog signal for its simulation, and it generates other output signal. Fig. 3 shows the need of the data conversion between the models. In a hybrid simulation, an output of a discrete event model becomes an input of a continuous model, and vice versa.

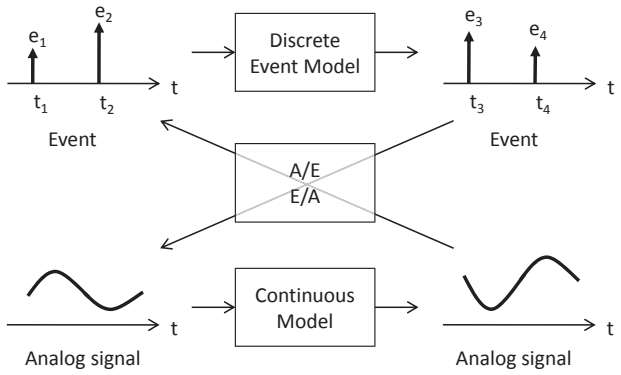


Fig. 3. Why is the data conversion necessary?

For this data conversion, we use an analog-to-event (A/E) and event-to-analog (E/A) converter [11]. The E/A conversion is conducted as shown in the event-function mapping table shown in Fig. 4. The table defines how the discrete events are converted to continuous signals. The signal is a function of time. On the other hands, A/E conversion is conducted by zero-crossing detection [14]. An event is triggered when a continuous signal reaches the threshold level, and the event message is determined by the hit crossing direction.

IV. Proposed Framework for Simulation of Hybrid Systems

From a software engineering perspective, the software quality has to be considered for the modeling and simulation of

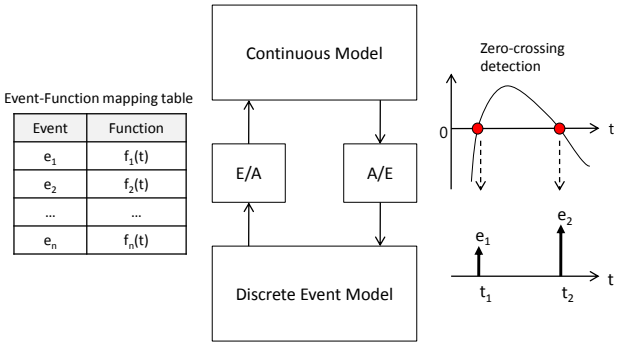


Fig. 4. Data conversion using A/E and E/A converter

hybrid systems. The new requirements for software quality are *reusability*, *maintainability*, and *interoperability* [15]. In order to satisfy the requirements, we propose an interoperation framework for hybrid systems using HLA/RTI. Our proposed framework has several assumptions about simulation environment as follows:

1) A MATLAB/Simulink model is used for the modeling and simulation of continuous systems. MATLAB is a representative mathematical computing tool that provides a flexible environment for continuous systems modeling and simulation.
2) Lookahead = 0. (This means that the models are simulated without delay.)
3) Mutually exclusive activation between DES and CS. In our approach, DES acts as a controller for CS in such matters as the relation between a controller and a plant.

A. Framework Design and Implementation

A discrete event model and a continuous model are simulated in their own ways, as explained in Section III. In addition, the simulators may be developed in different environments. To interoperate different simulators without any modification, we use an HLA interface, a so-called *HLA adaptor*, to connect the simulators and RTI. The simulator requires essential services and callbacks to simulate itself, and the HLA adaptor helps the modelers to select the necessary interfaces easily.

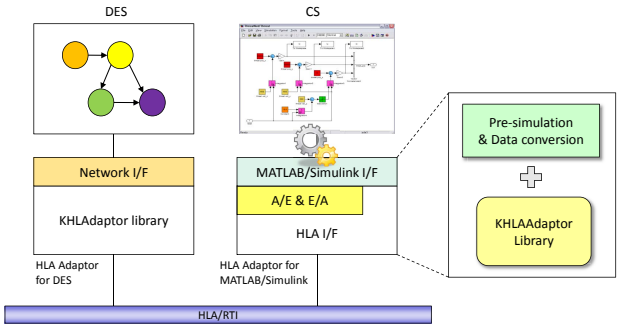


Fig. 5. Proposed framework for simulation interoperability of hybrid systems

Fig. 5 shows the proposed framework for the interoperation of the hybrid system models. A discrete event model uses

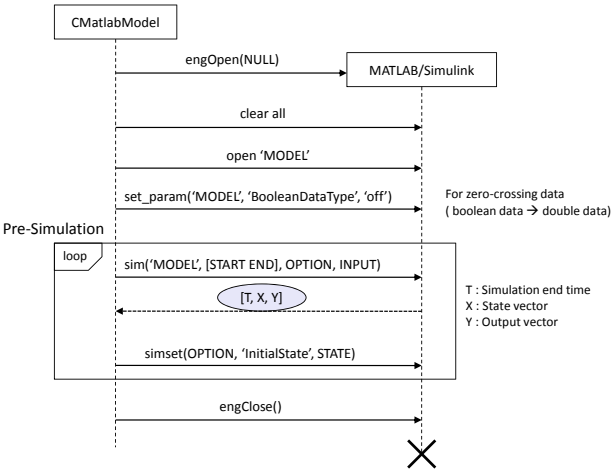


Fig. 6. Implementation of the MATLAB/Simulink interface in an HLA adaptor

an adaptor for HLA, and the adaptor is developed using a KHLAAdaptor library [16]. On the other hand, the existing adaptor library cannot be used for a continuous model, because the time advancement of a continuous model needs pre-simulation, as explained in the previous section. Therefore, an adaptor for a continuous model needs modification of the existing KHLAAdaptor library, and it includes pre-simulation algorithm and data conversion rules. To support the functions, a *CMatlabModel* class is added in the KHLAAdaptor library. The role of the class is depicted in Fig. 6. Using the class, the adaptor opens a continuous model developed using the MATLAB/Simulink, and controls the model. For pre-simulation, the adaptor executes the model during the time window, from *START* to *END* time, using input signal data, and then the return values are used for next pre-simulation. The functions for the MATLAB/Simulink interface in Fig. 6 are explained in Table I. The *sim* and *simset* functions are used for the pre-simulation of a continuous model.

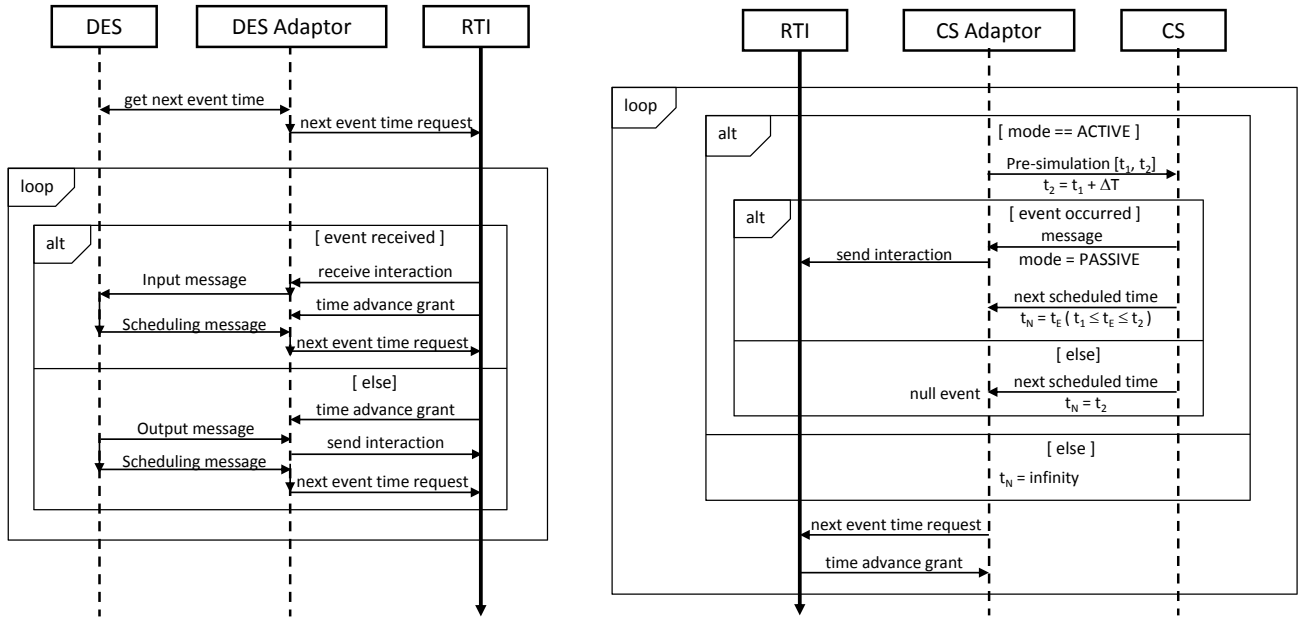
TABLE I
FUNCTIONS FOR MATLAB/SIMULINK INTERFACE

Function	Description
engOpen	State MATLAB process
open	Open Simulink model
set_param	Set simulation parameters
sim	Simulate a model with a specific input data during [start time, end time]
simset	Edit simulation parameters
engClose	Quit MATLAB process

B. Simulation of the Framework

The simulation of the proposed framework is depicted in Fig. 7. An adaptor for a discrete event model checks the next-event time from the model and requests the time to RTI. The next time is deterministic because the scheduling time is predictable in the discrete event model. If an event is received, *receiveInteraction* API of HLA is called, and the message is transferred to the discrete event model. After it calculates the message, the discrete event model is re-scheduled, and the adaptor requests the new scheduling time to RTI. If no event is received, the model generates an output message, and the adaptor sends the message to the continuous model through RTI.

An adaptor for a continuous model conducts the pre-simulation algorithm and data conversion. The adaptor controls the continuous model according to the *TIME-ADVANCE* procedure, as shown in Algorithm 2. In *ACTIVE* state, the



(a) Sequence of DES

(b) Sequence of CS considering pre-simulation

Fig. 7. Simulation of the proposed framework

adaptor executes the continuous model during $[t_1, t_1 + \Delta T]$. The pre-simulation returns information about a state event or a time event. If a state event occurs at the threshold point, the adaptor sends the event to the discrete event model through the *sendInteraction* API of RTI. The state of the adaptor changes to *PASSIVE*, and the next scheduled time is set to the event time, t_E. If no event has occurred during the pre-simulation, a time event occurs at the end of the pre-simulation. The next scheduling time is set to $t_1 + \Delta T$. The adaptor requests the next-event time, t_E of $t_1 + \Delta T$, and this process is repeated. When a discrete event is received by the adaptor for the continuous model, the event converts into a continuous signal according to an event-function mapping table. The converted signal is used for an input of the *sim* function of a *CMatlabModel* class. An adaptor for a continuous model is, as it were, a state event finder and a data converter as well as an agent to connect with HLA/RTI. The time management of HLA supports synchronization between both adaptors.

V. Experimentation

The proposed framework is applied to the simulation of a water level control system. In Fig. 8, a water level control system consists of two subsystems, a controller and a water tank. In the water tank, there are two water level sensors (*overflow, shortage*), and the detected state of the water tank is transmitted to the controller. The controller uses an ON/OFF switch to control the water tank, and the control signal controls the operation of the tank.

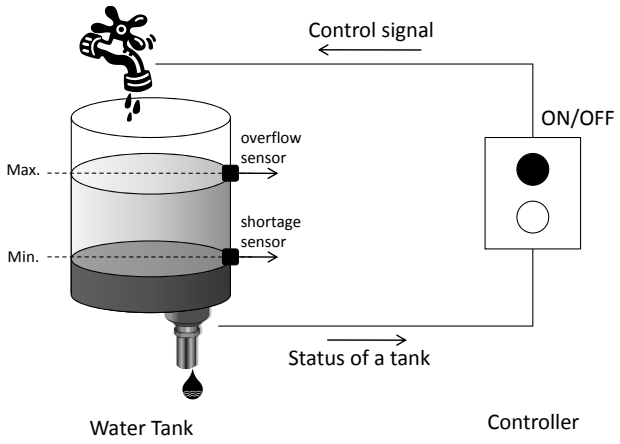


Fig. 8. Water level control example

A. Continuous Model

Fig. 9 shows water entering the tank from the top and leaving through an orifice in its base. The rate at which the water enters is proportional to the voltage, V, applied to the pump. The rate at which the water leaves is proportional to the square root of the height of the water in the tank. The differential equation for the height of water in the tank, H, is given in equation 1.

$$A\frac{dH}{dt} = bV - a\sqrt{H} \tag{1}$$

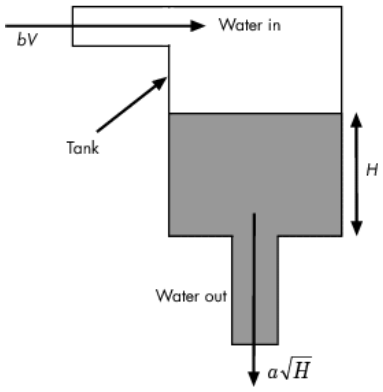


Fig. 9. Water tank example

, where

V : input voltage of water tank
A : cross-sectional area of a tank
b : a constant related to the flow rate into a tank
a : a constant related to the flow rate out of a tank

The mathematical form of the water tank model is described using a MATLAB/Simulink as shown in Fig. 10. The model contains one state, H, one input, V, and one output, H. The height of the water tank is the state of the equation, and the value of the state is changed via continuous time. The input of the model is a voltage signal, and the value determines the quantity entering the water tank.

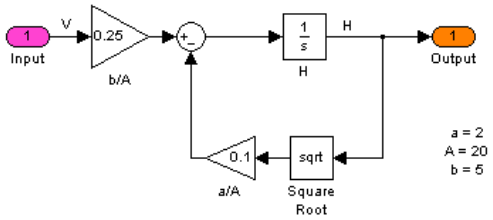


Fig. 10. Simulink model of water tank (CS)

B. Discrete Event Model

A controller is a discrete event system that controls the volume of water entering a tank. This system is modeled using DEVS formalism, and the model is shown in Fig. 11. The initial state of the model is *WAIT*, and the time advance of the state is *infinity*. When the model receives a *shortage* event from the water tank, it changes the state from *WAIT* to *ON*. Because the event means that the tank is lacking water, an *on* event occurs to control the entering water. On the other hand, when an *overflow* event is received, the state is changed to *OFF*, and then an *off* event occurs.

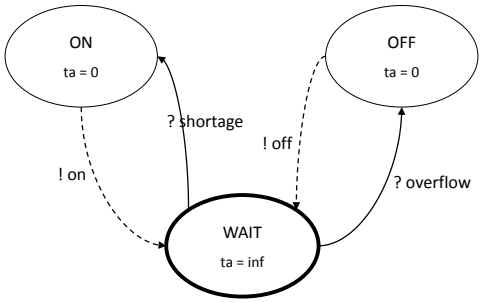


Fig. 11. DEVS model of water control (DES)

C. Data Conversion

Data conversion is necessary for co-simulation between simulators with different characteristics. A water tank needs analog signals to operate a pump, and it has continuous values represented by the height of the water in the tank. In contrast, a controller requires events for a state transition.

The continuous values for the amount of the water in the tank are detected by threshold sensors. Fig. 12 describes the relation between threshold values and events. When the height of the water reaches *4* because the water is rising, an $overflow$ event occurs. Likewise, a $shortage$ event occurs when the height reaches *1* because the water is falling. The events are used for inputs to the controller. We used the hit crossing block of the MATLAB/Simulink to convert the analog signal into the discrete event.

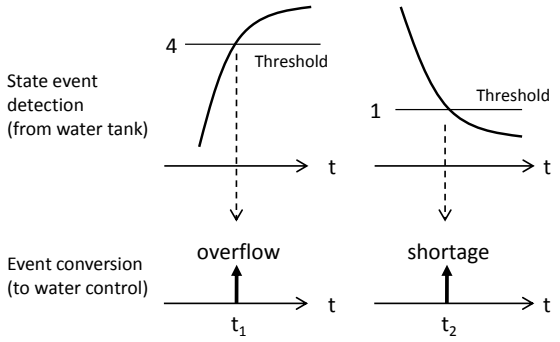


Fig. 12. A/E conversion: state event detection from a water tank model

When events occur from a water level controller, they have an effect on the operation of the water tank. The events are converted to a segment function, in that the water tank is a continuous model. The conversion relation is shown in Table II. An event $on(t = t_1)$ is transformed into $f_1(t) = u(t - t_1)$ which means that input value of a water tank is one for a positive argument, i.e. after t_1 time. Likewise, an event $off(t = t_2)$ is transformed into $f_2(t) = u(t_2 - t)$.

TABLE II
EVENT-ANALOG CONVERSION OF WATER LEVEL SYSTEM

Event	Analog
$on(t = t_1)$	$f_1(t) = u(t - t_1)$
$off(t = t_2)$	$f_2(t) = u(t_2 - t)$

D. Federation Design and Implementation

In order to simulate a controller and a water tank model with RTI, an HLA interface should be implemented for each simulator. The HLA interface is a form of adaptor, as explained in Section IV. In the case of the controller, the design method of its adaptor is shown very well in [17]. The design and implementation of the HLA adaptor for the water tank as a continuous model is conducted using an adaptor for the MATLAB/Simulink model described in the previous section.

```
(OMDT v1.3.5.17)
(ObjectModel (Name " ")
             (VersionNumber " ")
             (Type FOM)
             (ModificationDate 11/29/2010)
             (MOMVersion "1.3")
             (FEDname "FederationName")
  (EnumeratedDataType (Name "CONTROL_COMMAND")
        (AutoSequence No)
        (StartValue 1)
        (Enumeration (Enumerator "ON")
                     (Representation 1))
        (Enumeration (Enumerator "OFF")
                     (Representation 2))
  )
  (Interaction (ID 1)
     (Name "CONTROL")
     (ISRType IR)
     (DeliveryCategory "reliable")
     (MessageOrdering "timestamp")
     (Parameter (Name "ID")
                (DataType "unsigned long")
                (Cardinality "1")
                (Accuracy "perfect")
                (AccuracyCondition "always")
     )
     (Parameter (Name "COMMAND")
                (DataType "CONTROL_COMMAND")
                (Cardinality "1")
                (Accuracy "perfect")
                (AccuracyCondition "always")
     )
  )
)
```

Fig. 13. Common data for simulation of water level system

The common data between simulators, the so-called Federation Object Model (FOM), includes one interaction, and it has one parameter, as shown in Fig. 13. The interoperability data is a discrete event, ON or OFF. The event is converted to the segment function defined in Table II in the HLA adaptor for a continuous model.

In the initialization stage, input ports and hit crossing points are registered in the HLA adaptor. The initial signal of the input port $control$ is 1, which denotes a positive value of $u(t)$. In the E/A conversion stage, the input signal function is determined by input events. When an input event is OFF, the function of the input port $control$ is set to 0, which means a negative value for $u(t_1 - t)$, and t_1 is event occurrence time. In the case of ON, the function is 1, i.e., $u(t - t_2)$.

The overall architecture of the water level control system is shown in Fig. 14. The controller model is implemented using DEVSim++ [18], which is C++ library to support implementation of the DEVS models, and the water tank model is a form of the MATLAB/Simulink model. Two models are connected to their adaptors, which support the HLA service using MÄK RTI 3.4.

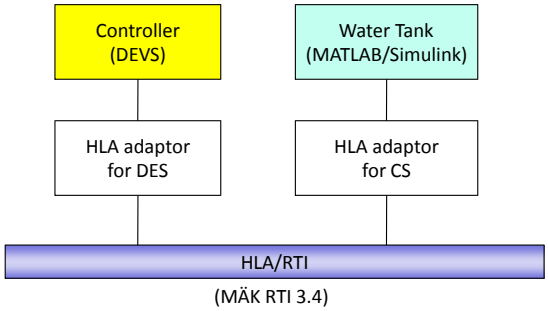


Fig. 14. Overall architecture of water level control system

E. Simulation Result

Fig. 15 shows a simulation result. The water fills the tank when the *ON* switch of a controller is turned on. When an overflow sensor, which is located at height 4, detects the water, the height of the water in the tank goes down. In contrast, when the water decreases to height 1, a shortage sensor is activated, and the height of the water begins to rise. The controller and water tank model were not modified for simulation interoperability, and the proposed framework enables reuse of the simulators.

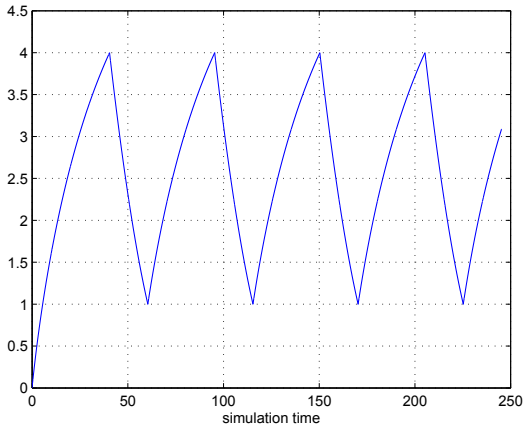


Fig. 15. Variation of water height

VI. Conclusions and Future Works

This paper proposes an interoperability framework for the simulation of hybrid systems. A hybrid system is a mixed form of discrete events and continuous systems. Each system has different characteristics, and the models are developed in different environments. Our adaptor-based interoperation framework using HLA/RTI makes it possible to reuse existing models and to interoperate with other simulators. The adaptor includes an HLA interface, pre-simulation algorithms, and data conversion. The water level experiment shows the hybrid simulation using an HLA adaptor, and we could reuse the existing models without any modification. Further studies are needed to simulate discrete event models and general continuous models in addition to MATLAB/Simulink models. Moreover, faster and more efficient pre-simulation algorithms should be discussed.

Acknowledgment

This work was supported by Defense Acquisition Program Administration and Agency for Defense Development under the contract UD080042AD, Korea.

References

[1] "IEEE standard for modeling and simulation (M&S) high level architecture (HLA) - framework and rules," *IEEE Std 1516-2000*, pp. i–22, Sep. 2000.

[2] "IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification," *IEEE Std 1516.1-2000*, pp. i–467, 2001.

[3] "IEEE standard for modeling and simulation (M&S) high level architecture (HLA)-object model template (OMT) specification," *IEEE Std 1516.2-2000*, pp. i–130, 2001.

[4] J. S. Dahmann, R. M. Fujimoto, and R. M. Weatherly, "The Department of Defense High Level Architecture," in *Proceedings of the 29th conference on Winter simulation*, ser. WSC '97, Atlanta, GA, United States, Dec. 1997, pp. 142–149.

[5] H. Zheng, "Operational semantics of hybrid systems," Ph.D. dissertation, Berkeley, CA, USA, 2007, adviser-Edward A. Lee.

[6] A. Borshchev, Y. Karpov, and V. Kharitonov, "Distributed simulation of hybrid systems with AnyLogic and HLA," *Future Gener. Comput. Syst.*, vol. 18, no. 6, pp. 829–839, 2002.

[7] E. Kofman, M. Lapadula, and E. Pagliero, "PowerDEVS: A DEVS-Based Environment for Hybrid System Modeling and Simulation," School of Electronic Engineering, Universidad Nacional de Rosario, Tech. Rep. LSD0306, 2003.

[8] M. Clune, P. Mosterman, and C. Cassandras, "Discrete Event and Hybrid System Simulation with SimEvents," in *Discrete Event Systems, 2006 8th International Workshop on*, Ann Arbor, MI, USA, Jul. 2006, pp. 386–387.

[9] F. Bouchhima, G. Nicolescu, E. M. Aboulhamid, and M. Abid, "Generic discrete-continuous simulation model for accurate validation in heterogeneous systems design," *Microelectron. J.*, vol. 38, no. 6-7, pp. 805–815, 2007.

[10] M. Wetter and P. Haves, "A Modular Building Controls Virtual Test Bed for The Integration of Heterogeneous Systems," in *SimBuild 2008 - 3rd National Conference of IBPSA-USA*, Berkeley, CA, USA, Jul. 2008, pp. 69–76.

[11] S. Y. Lim and T. G. Kim, "Hybrid Modeling and Simulation Methodology based on DEVS Formalism," in *SCSC '2001*, Orlando, FL, USA, Jul. 2001, pp. 188–193.

[12] J. F. Klingener, "Programming combined discrete-continuous simulation models for performance," in *WSC '96: Proceedings of the 28th conference on Winter simulation*, Coronado, CA, USA, Dec. 1996, pp. 833–839.

[13] F. E. Cellier, "Combined continuous/discrete system simulation by use of digital computers: techniques and tools," Ph.D. dissertation, Zürich, Switzerland, 1979.

[14] P. A. Fishwick, *Handbook of Dynamic System Modeling (Cpaman & Hall/Crc Computer and Information Science)*. Chapman & Hall/CRC, 2007.

[15] J. P. Cavano and J. A. McCall, "A framework for the measurement of software quality," in *Proceedings of the software quality assurance workshop on Functional and performance issues*, 1978, pp. 133–139.

[16] J.-H. Kim, S.-Y. Hong, and T. G. Kim, "Design and Implementation of Simulators Interoperation Layer for DEVS Simulator," in *M&S-MTSA'06*, Calgary, Canada, Jul. 2006, pp. 195–199.

[17] T. G. Kim, C. H. Sung, S.-Y. Hong, J. H. Hong, C. B. Choi, J. H. Kim, K. M. Seo, and J. W. Bae, "DEVSim++ Toolset for Defense Modeling and Simulation and Interoperation," *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology*.

[18] T. G. Kim, *DEVSimHLA User's Manual*, 2007. [Online]. Available: http://smslab.kaist.ac.kr

An Analysis of the Cost of Validating Semantic Composability

Claudia Szabo and Yong Meng Teo
Department of Computer Science
National University of Singapore
Computing 1, 13 Computing Drive
Singapore 117417
Email: claudias, teoym@comp.nus.edu.sg

***Abstract*—Validation of semantic composability is a non-trivial problem and a key step in component-based modeling and simulation. Recent work in semantic composability validation promise to reduce verification, validation, and accreditation efforts. However, the underlying cost of current validation approaches can undermine the promised benefits, and the trade-off between validation accuracy and validation cost is not well understood. In this paper we present, to the best of our knowledge, the first quantitative study on the cost of validating semantic composability. Firstly, validation approaches are categorized into techniques that validate general model properties, and techniques that validate model execution. Secondly, we focus on two key factors that influence validation cost, namely, simulation problem characteristics and the validation approach adopted. Our study covers four representative validation approaches, CD++ DEVS, Z-based DEVS, Petty and Weisel formal validation, and deny-validity, and for simplicity, we use computation time as a measure of validation cost. Based on a queueing network model with 1,000 components, the cost of validating general model properties accounts for 55% of total validation time, with the remaining 45% incurred by model execution validation. With respect to composition structure, a 10% increase in the fork and join component interconnections increases validation cost by more than half. In model execution validation, the time-based deny-validity approach is seven times more expensive than the timeless Petty and Weisel formalism.**

I. Introduction

In component-based simulation model development, components are combined and re-combined to meet user requirements [1]. *Syntactic verification* and *semantic validation* are two key steps in component-based simulation model development [2]. In syntactic composability, components have to be correctly connected and must interoperate. In semantic composability, the composition of all components must be meaningful, in terms of data exchange and context. Furthermore, the composition must be valid [2]. Model Verification, Validation and Accreditation (VV&A) is one of the most important steps in the simulation life-cycle. Classical non component-based validation methods include various stages and processes, and are often costly and time-consuming [3], [4]. The validation cost is further aggravated in component-based modeling and simulation where the complexity of the models increases rapidly with model size [2], [5].

The validation of semantic composability remains a non-trivial problem and its cost is not well understood. Firstly, composition is not a closed operation since valid components do not necessary form valid compositions [3]. Secondly, reused components are developed for different purposes and when composed may result in emergent properties [5], [6], [7]. Thirdly, there exist various validation perspectives on the component interactions over time, such as *model property* aspects including deadlock, safety, and liveness and *formal* measures of the validity of compositions, also called "figures of merit" [1]. Lastly, the composed model must be similar or close to the real system it abstracts [3]. Very often, the implementation of an automated process to evaluate this similarity is difficult, if not impossible [3], [5].

In recent years, several approaches have been proposed for the verification and validation of composed simulation models, most of which target the validation of semantic composability [2], [8], [9], [10]. In this paper, we have selected for evaluation a Z-based specification for validating Discrete Event System Specification (DEVS) models [9], an automated input/output transformation validation for CD++ and DEVS models [10], a formal theory of composability [2], and our proposed deny-validity approach [11]. These approaches focus on various aspects of validity, with several underlying assumptions and limitations that influence their outcome and applicability. An important issue is the computational cost of validation, especially since in the context of large-scale composed simulation models this cost increases exponentially [11], [12]. Simulation components are complex entities with a variety of attributes and complex behavior, and their composition often leads to state space explosion. In particular, the validation of the entire model space as proposed by model checking or theorem proving approaches [9] becomes unfeasible for models with a large number of components, connections, and/or interactions. The cost of comparing the composed model with a reference model grows exponentially with the number of components and their attributes and states. Besides problem size, other factors can influence the computational cost of semantic validation. These factors can be classified in two main categories: (i) simulation problem characteristics such as the number of components, the composition structure, and (ii) validation approach characteristics such as the level of abstraction, validation accuracy. We discuss these factors in the following sections.

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

To distinguish between existing validation approaches that can be employed in the simulation life-cycle, a common metric to evaluate the cost of validation is needed [13]. For simplicity, we propose in this paper to quantify the *computational cost* of semantic validation. We first analyze the validation cost of selected approaches in terms of simulation problem characteristics such as the number of components and composition structure. Next, we study the trade-off between validation accuracy and computational cost. Our main contribution is a comprehensive, quantitative analysis on the computational cost of semantic validation in terms of problem characteristics and validation accuracy.

This paper is organized as follows. Section II outlines the key factors that influence the cost of semantic validation. Section III presents our cost analysis covering the cost of validating model properties and model execution. We compare and contrast our work with existing evaluation studies in Section IV. Section V concludes this paper and discusses future work.

II. Cost of Semantic Composability Validation

Among the main factors that influence the computational cost of semantic validation are simulation problem characteristics, and validation approach. From a component-based perspective, measurable *simulation problem characteristics* include the number of components, the description of the components, the composition structure, and the degree of interaction between components. The *number of components* in the composed model describes the size of a component-based simulation model. More fine-grained measures of the composition size are determined by the way in which components are described, in terms of the *number of attributes*, and the *number of states* per component. The number of time delay attributes is also important, depending on the validation approach. The *composition structure* also influences the computational cost of validation. If a component-port paradigm is adopted, such as in DEVS [10], CoDES [14], or OSA [15], the composition structure is described in terms of the *number of connectors*. A large number of connectors, in particular those with complex semantics such as fork/join, can lead to state-space explosion even for composed models with a small number of components [11]. Another factor is the degree of *component interaction*, expressed in terms of the number of events executed per unit time. Validation approach characteristics that influence computational cost include validation techniques and validation abstraction. *Validation techniques* can have a major influence on the computational cost of validation. For example model checking techniques are known to be unfeasible for the validation of large models [12]. The *validation abstraction* has the potential to reduce the computational cost. For example, if time is not considered as in the Petty & Weisel formalism, cost can be reduced, as we will show in Section III.

We further analyze the cost of validating model execution, in which the composed model is compared with a reference model. The outcome of this evaluation depends on the formalism used to abstract the two models to facilitate reasoning. We analyze the influence of a time-based and a timeless formalism on computational cost and discuss the variation of cost based on several factors such as number of components and component interaction. Another important factor is the validation window size, defined as the interval during which the comparison between the composed model and the reference model is performed. Intuitively, given a single validation window, a larger window size ensures that deviant behavior of the composed model from the reference model can be identified. However, we have found that an increase in size of the validation window beyond a certain knee value comes at very high computational cost. This is similar with the relationship between accuracy and cost suggested in [16], in which after a certain knee value, increases in accuracy come at disproportionately larger increases in cost.

III. Cost Analysis

In this section we propose to evaluate the computational cost of four validation approaches, namely, the CD++ DEVS-based validation approach [10], the DEVS-based Z specification validation approach [9], the Petty & Weisel formal validation approach [2], and our deny-validity approach [11]. Our selection was based strictly on the availability of the code, or in the case whereby the code was not available, on the completeness of the descriptions in the published papers to facilitate faithful implementation. The Z specification language has been proposed to formally validate models represented in the DEVS formalism [9]. The atomic DEVS model is represented in Z in a time-less manner, and a theorem proving tool based on Z such as Z/EVES is used to verify the model and prove certain properties. However, the Z specification language limits the applicability of this approach to coupled DEVS models [9]. Wainer et al. proposed an input/output transformation validation tool that inputs certain data in the DEVS coupled model and expects specific data through an output point [10]. This lightweight approach treats a DEVS coupled model as a blackbox and is easy to use since it employs two well-known DEVS/CD++ constructs, i.e. *Generator* to generate input, and *Acceptor* to accept input. However, only a single output point can be tested and only primitive data types such as real and integer are considered. In the formal theory of semantic composability validation proposed by Petty and Weisel a composed simulation model is modeled as the composition of mathematical functions that represent components over one-dimensional integer domains [2]. The simulation of the composed model is represented as a Labelled Transition System (LTS) [17], where nodes are model states, edges are function executions, and labels are model inputs. Using several metrics, the distance between the composed model and a reference model is calculated. However, time is not modeled and the approach is not feasible for compositions with feedback loops and fork and join connectors. Moreover, the reference model is assumed to exist a-priori.

In our previous work, we proposed a deny-validity approach in which the composed model was subjected to a battery

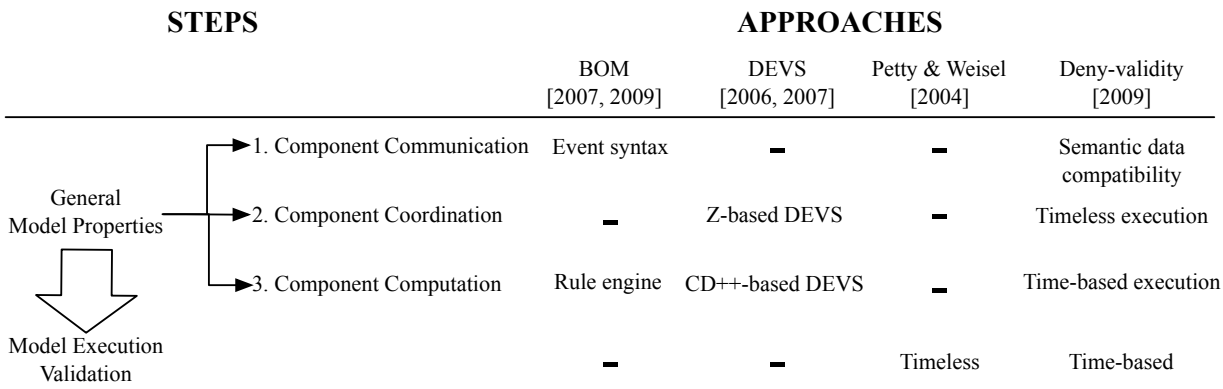


Fig. 1. Landscape of Recent Validation Approaches

of tests aimed at discarding it as semantically invalid[1] [11]. Informally, we attempt first to eliminate models in which components cannot communicate and coordinate meaningfully. While properties such as communication and coordination fall into the general definition of simulation verification [3], they are included in the definition of semantic composability [5], [2] and as such they are validated in this layer. Next, models with invalid semantic composability are also those that have valid model properties, but whose execution is not close to that of the real system the composed model abstracts [3]. To eliminate models that are not similar to the real system being abstracted, we compare between the composed model and a reference model. In contrast to current static perfect model validation [2], our proposed *time-based formalism* represents dynamic component behavior, can represent fork and join connectors, and is applicable to a composed models with wide variety of topologies and interactions. Furthermore, we are able to quantify the similarity between the composed model execution and the reference model execution using our defined semantic metric relation.

We divide our evaluation process in two stages. Firstly, we evaluate the cost of validating general model properties. Secondly, we evaluate the cost of comparing between the composed model and a reference model. The factors that influence the computational cost of validation form a complex parameter space in which the influence of each individual factor is difficult to identify. Thus for simplicity, we employ a single-server queue model with a varying number of components. A more complex example is analyzed in Section III-D.

A. Evaluation Methodology

To better understand the landscape of semantic composability validation, we summarize recent validation approaches in Fig. 1. An important observation is that most component-based verification and validation approaches focus on two main aspects, namely, on the validation of general model properties, and the model execution by the comparison of the composed and the reference models. General model properties are in turn divided into input/output transformations, component coordination, and component communication. Current approaches such as the CD++ DEVS-based approach, the DEVS-based Z specification approach perform a subset of these validations. Comparison between the composed model and a reference model is performed by the Petty & Weisel approach. Lastly, our deny-validity approach targets both validation stages.

For our study, we were able to obtain the code for the CD++ DEVS-based approach. We implemented the second DEVS-based approach following the complete descriptions given in [9]. However, the Z specification does not permit the representation of coupled or connected objects, and as such it cannot be used to specify component-based models. Instead, we employed the Object-Z formalism [18], which can be easily adapted in a similar manner as that suggested in [9], to cater for DEVS coupled models. However, model checking based on Object-Z is still in its early stages. Nonetheless, we were able to reach the outcome described in [9], i.e. syntax, type, and inconsistency checking by employing the Wizard checker for type and syntax checking [18]. Ongoing work exists to include the Object-Z specification into theorem proving tools such as Isabelle/HOL [18], but these were beyond the scope of our study. The advantage and major improvement of our implementation is that it caters for component-based models. We implemented the Petty & Weisel approach following details from the published papers [19], [2]. A fundamental assumption in the Petty & Weisel approach is that a simulation component can be transformed into a mathematical function over integer domains. Nonetheless, no details about how this transformation should be performed are provided. This transformation is crucial when computing the mathematical composability of the functions that represent components, which by definition translates to verifying integer domain inclusion. In our implementation, we employed a brute-force transformation by mapping every component output into an integer number (iteratively for each output) and assuming that the functions are mathematically composable. This is not a limitation in our study of computational cost because checking for mathematical composability is not a major component of the validation cost in the Petty & Weisel approach[2].

[1]We ensured that models with invalid syntax were eliminated using an approach based on compositional grammars expressed in EBNF [14].

[2]In particular, provided that the transformation from the meta-model to integers is sound, the algorithm to determine mathematical composability could employ a Radix sort algorithm to first sort the integer values, followed by an inclusion check, resulting in $O(kn)$ steps, where n is the number of elements in the interval, and k is the average element length.

Our study follows the two main validation steps described in Fig. 1. Firstly, we evaluate the computational cost of approaches that aim to validate general model properties, namely, the CD++ DEVS-based validation approach [10], the DEVS-based Z specification validation approach [9], and our deny-validity approach [11]. We employ a simple queue model with varying number of service unit components as shown in Fig. 2.

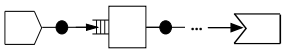


Fig. 2. Simple Queueing Model

We have chosen this queue model because of its relative simplicity with respect to factors such as the number of attributes per component and composition structure. Besides the need for simplicity, we have also excluded these factors because they are not considered by most validation approaches. Secondly, we study two approaches that compare the composed model with a reference model, namely, Petty & Weisel's validation approach [19] and our deny-validity approach. We evaluate the variation of cost with model size and analyze these approaches in terms of accuracy and cost. We first show the impact of timeless abstractions, such as that employed by Petty & Weisel, on the validation process. Next, we evaluate the variation of cost with the validation accuracy, expressed using the size of the validation window during which the composed model is compared with the reference model. We present an average of ten execution runs and execute all experiments on an Dell PowerEdge SC1430 Dual Quad Core Server, with Intel Xeon, 1.83 GHz, and 4GB RAM.

B. General Model Properties

The validation of general model properties focuses on three main properties, namely, component communication (P1), component coordination (P2), and component computation (P3), of the composed model. The validation of component communication aims to check data compatibility between connected components in the composition. Current approaches ensure that components follow a common reference model [20], verify syntax in terms of event parameters and data types [21], or validate semantic compatibility using a component-based ontology [14]. The validation of component coordination aims to check that interleaved executions of components in the model are correct [4]. This is usually done using model checkers, and in general by abstracting time in instantaneous transitions [12], [11]. Lastly, the validation of component computation aims to check that the components can execute during the composition run. For simplicity, the Z-specification DEVS-based and the CD++ approach will be hereafter referred to as DEVS1 and DEVS2 respectively.

Table I presents the variation of computational cost when varying the number of components from 10 to 1,000. While the computational cost increases proportionally with the composed model size, the increase is insignificant in the DEVS1 and DEVS2 approaches, in which the computational cost

#Comp	Runtime (s)					
	DEVS1	DEVS2	Deny-validity			
	P2	P3	P1	P2	P3	Total
10	< 0.1	0.2	< 0.1	5.3	51.1	56.5
100	< 0.1	0.5	0.2	146.6	43.5	190.3
500	0.2	4.5	0.3	193.6	67.0	260.9
1,000	0.7	16.7	0.5	330.4	130.9	461.7

TABLE I
COMPUTATIONAL COST OF VALIDATING GENERAL MODEL PROPERTIES

approaches one second and 17 seconds for 1,000 components respectively. In contrast, the deny-validity approach takes close to 7 minutes for the same model. The reason for this discrepancy becomes evident when we look at the cost components of the deny-validity approach, which is the only approach that validates all general model properties. The cost of validating property P1 is insignificant for this model, but increases with the number of data types that a component outputs. This is because P1 is validated by establishing data compatibility between components using our proposed ontology, which is queried for every pair of connected components [14]. Space constraints prevent us from showing these results here. The validation of property P2 incurs the largest cost of the three properties. The validation is done using the SPIN [22] model checker to validate a specification of the composed model, and becomes highly unfeasible when the composed model size increases beyond 250 components. As such, flags that limit the search space are employed for validation of models with 500 and 1,000 components. In particular, we use the "`-w`" flag to reduce the depth of the search tree, and "`DMEMLIM`" to cap the amount of memory used. Property P2, component coordination, is also validated by the DEVS1 approach. However, this approach focuses only on type and syntax checking, and does not look at all possible interleaved execution states, like the deny-validity approach.

The validation of property P3 is performed in a similar manner for DEVS2 and the deny-validity approach, by checking several types of data at a connection point. However, the DEVS2 approach can only process a limited number of events and requires the user to input the exact moment in time when the output is expected. Moreover, the DEVS2 approach validates input at the last connection point in the composition. Lastly, only primitive data types such as real or integer can be validated. In contrast, in the deny-validity approach the user can specify desired data of any type, including domain specific, at any connection point in the composition. Additional liveness properties are also validated [11].

C. Model Execution Validation

The validation of model execution by comparing the simulation model with a real or referent system is a traditional validation approach [3]. This comparison is done in component-based simulation by the Petty & Weisel [2] and the deny-validity approaches [11]. These two approaches follow a similar validation sequence, which can be separated into three major steps: (i) transformation of components into formalism; (ii) transformation of composition formalism into a

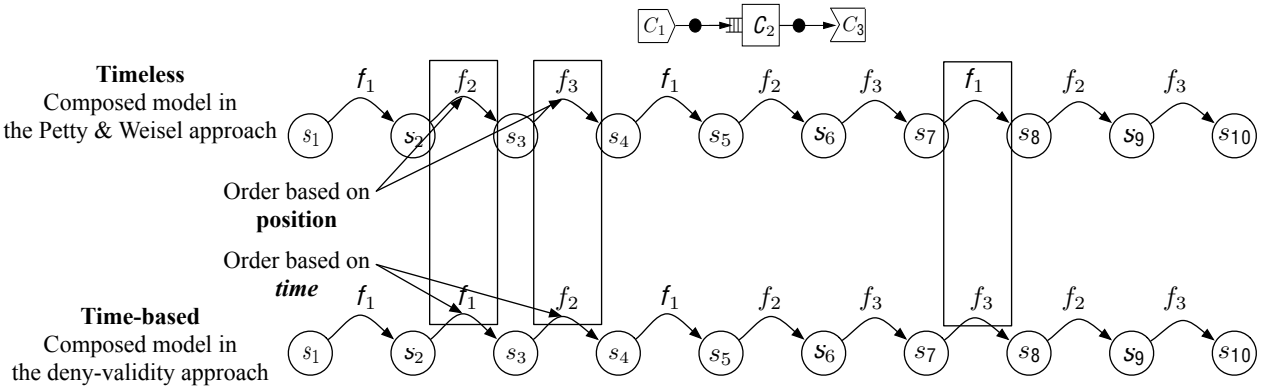


Fig. 3. Execution Order: Timeless vs Time-based Validation

Labeled Transition System (LTS) [17]; and (iii) comparison with reference model. Both approaches rely on a formal comparison between the simulation of the composed model and that of a reference model. The major difference between the two approaches lies in that Petty & Weisel employ a static formalism, in which a component is represented as a function over integer domains, whereas the deny-validity approach proposes a time-based formalism in which a component is represented as a function over a three-coordinate domain containing time, state, and input/output. The Petty & Weisel approach offers a high level of abstraction, which permits reasoning about closeness under composition and has reduced computational cost as we will show below. However, it is difficult to transform component representations into integer values automatically, and the approach assumes that model properties, such as syntactic composability, safety, liveness, are validated beforehand. Moreover, the comparison process orders the component functions based on the location (left to right) of the components in the composition, which does not permit the validation of compositions with fork and join connectors and feedback loops. This is because the functions representing components on the fork/join branches need to be ordered during execution, which is not possible using only integer value domains. The same applies to feedback loops, where a mathematical composability ordering based on the position of the components cannot be deduced. Fig. 3 highlights the difference between timeless and time-based ordering. As it can be seen, the LTS that represents the composed model has an equal number of states in both approaches, but a different sequence of function execution. The deny-validity approach orders the functions based on the time moment when the component interacts with its neighbors, resulting in the sequence $f_1, f_1, f_2, \ldots$. In contrast, the Petty & Weisel approach orders the functions based on the position of the components in the composition, resulting in the sequence $f_1, f_2, f_3 \ldots$. Although there are instances where a position ordering is useful, e.g. when reasoning about validation closeness under composition, the latter time-based ordering paints a more realistic picture of the execution of the composed model. However, the time-based formalism employed in our deny-validity approach incurs an additional validation overhead.

The computational cost of validation can also be divided into three components, namely, (i) f, the *formalism cost*, as the cost of transforming from the component representation to the chosen formalism; (ii) p, the *process cost*, as the cost of transforming the composed model into a LTS using the chosen formalism; and (iii) c, the *comparison cost*, as the cost of comparing between the two LTS, representing the composed model and the reference model respectively. These cost components can be further refined as functions of various component and composition characteristics, as shown in Table II. An important point to highlight is that the cost of obtaining or constructing the reference model is not included. This is because the reference model is assumed to exist a-priori in the Petty & Weisel approach, whereas in our approach the reference model is automatically derived based on generic descriptions of reference components [11].

Cost Component	Petty & Weisel	Deny-validity
Formalism - f	$\geq f(n,\tau)$	$f(n,s,t,\tau)$
Process - p	$p(n,\tau)$	$p(n,s,t,\tau)$
Comparison - c	$c(n,\tau)$	$c(n,\tau) + c_{eps}(n,a,\tau)$

TABLE II
COST FACTORS

Our results presented below show that the number of components (n) drastically influences the computational cost. This is because the fundamental unit of each validation approach is the mathematical function, which represents a component. Other parameters, such as the average number of states per component (s) and the average number of attributes per component (a), influence the cost of the deny-validity approach but not the cost of the Petty & Weisel approach. This is because the Petty & Weisel approach deals only with integer representations. Nevertheless, the influence of the number of states and attributes on the computational cost in the Petty & Weisel approach should be more evident in the formalism cost, f, because the transformation from the component representation to unique and meaningful integer values should consider states and attributes as well. However, Petty & Weisel do not discuss details about how this transformation is performed. Another parameter that influences the computational cost is τ, the size of the validation window during which the composed model is compared to the reference model. This translates

into the number of simulation steps in the Petty & Weisel approach, and into the unfolding degree in our deny-validity approach. For example, for the simple model in Fig. 3, $\tau = 3$, resulting in three simulation steps and ten states for the LTS representing the composed model. The parameter τ can be seen as a measure of the accuracy of the validation process if we agree that the longer the interval under which the composition is observed as compared with the reference model, the more accurate is the validation result. We next evaluate the computational cost of semantic validation with the composition size in terms of the number of components, n and analyze the trade-off between the computational cost and the validation window size τ.

1) Results and Analysis: We implemented the Petty & Weisel approach based on its description [19], [2]. Since the details of mapping each component into a function over integer domains are not documented in Petty & Weisel approach, we used a simple and fast heuristic as discussed in Section III. Next, the composed model LTS was created as shown in Fig. 3. Lastly, we implemented the comparison between the composed model LTS and a reference model LTS using the BISIMULATOR tool [23], which is the same one we employ in comparing between the LTS of the composed model and reference model in the deny-validity approach.

Table III presents the variation of computational cost when varying the number of components from 10 to 1,000, with a validation window of size $\tau = 3$.

# Components	Runtime (s)	
	Petty & Weisel	Deny-validity
10	2.15	5.46
100	3.68	14.27
500	9.36	35.87
1,000	14.63	76.10

TABLE III
COMPARISON WITH REFERENCE MODEL

As it can be seen in Fig. 4, the computational cost of the Petty & Weisel approach is reduced, e.g. to less than 15 seconds for a simple queueing model with 1,000 components. In contrast, our deny-validity approach has a runtime of around one minute for the same model. This is because in our approach, a larger number of components implies that a larger number of component executions have to be ordered towards achieving the time-based ordering presented in Fig. 3.

To evaluate the trade-offs between computational cost and accuracy of the validation process, we evaluate the runtime cost of validating a queueing model with 1,000 components while varying the values of the validation window size τ to 3, 10, 15, and 20. Our results are presented in Fig. 5.

The increase of the computational cost with the validation window size τ is insignificant for the Petty & Weisel approach. As it can be seen in Fig. 5, for $\tau = 25$, the Petty & Weisel approach takes on average 40 seconds for a M/M/1 model with 1,000 components. In contrast, there is an evident trade-off in the deny-validity approach. Specifically, the validation

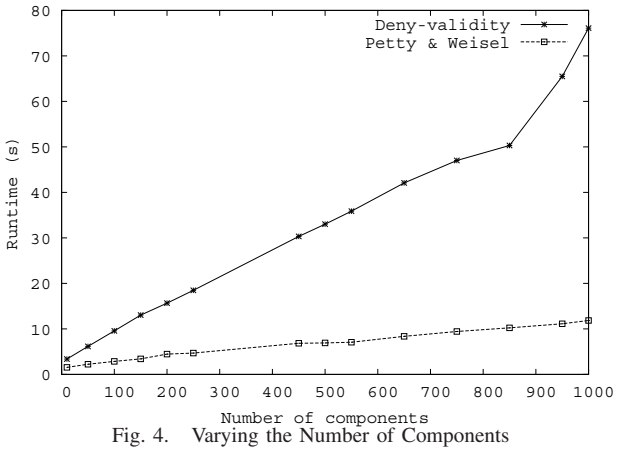


Fig. 4. Varying the Number of Components

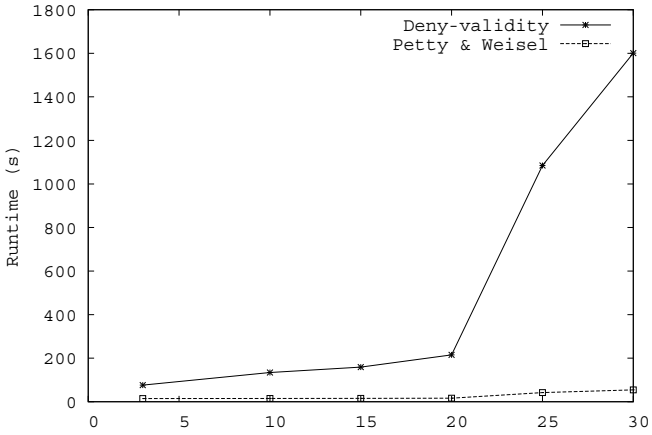


Fig. 5. Cost vs Accuracy

runtime increases from around 3 minutes for $\tau = 20$, to around 20 minutes for $\tau = 25$. The explanation for this decrease in performance is the following. Our validation process includes a time-ordering module which orders all functions based on the time moment in which components interact with neighbors. Thus a correct time ordering of components is necessary. The computation of this ordering is a constraint satisfaction problem, requiring a constraint solver to solve several equations. For $\tau = 25$, the LTS of the composed model has around 25,000 states. This translates into 25,000 constraints over the entire positive integer domain (since time values are integers) that have to be solved in order to determine the time ordering of the function executions, as required by our validation process. This operation has to be performed twice, for the composed and the reference model respectively. In solving these constraints, we employ the Choco constraint solver, which for our constraint types has a theoretical complexity of $O(c^3)$, where c is the number of constraints. This problem does not appear in the Petty & Weisel approach, because time is not modeled. For the curve in Fig. 5, we have calculated the knee values beyond which increases in validation window size come at disproportionate increases in validation cost. For models with 500 components, we have calculated the knee value as $\tau_{knee} = 15.22 \approx 15$. For models with 1,000 components, $\tau_{knee} = 18.99 \approx 19$.

D. Further Insight

In the above, we limited our evaluation study to only two factors, namely, the number of components n and the validation window size τ. Another important factor that influences the cost of validation is the composition structure, described by the number of one-to-one, fork, and join connectors. Existing validation approaches cannot validate models with fork and join connectors. However, this is possible in our deny-validity approach. We analyze the cost of validating composed models with fork and join connectors as shown in Fig. 6.

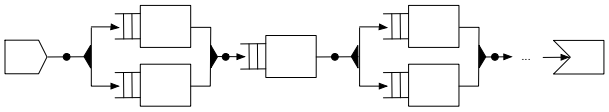
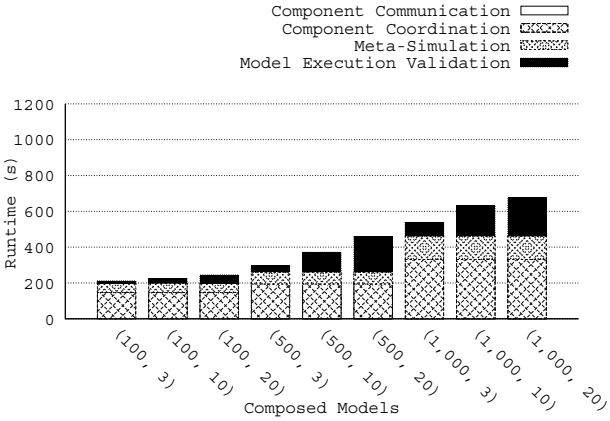
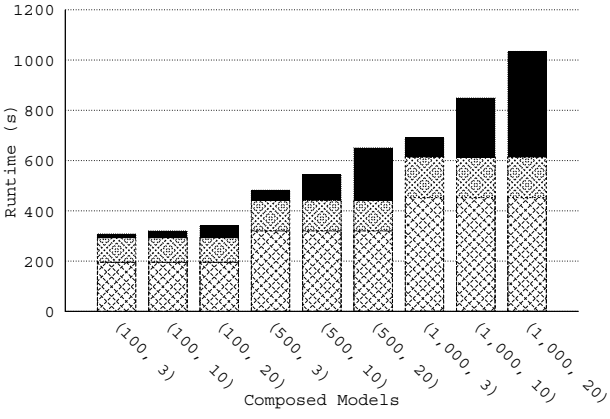


Fig. 6. Queueing Model with Fork/Join Connectors

Fig. 7(a) presents a breakdown of the validation cost for composed models with only one-to-one connectors. Fig. 7(b) presents a breakdown of the validation cost for composed models with a ratio $r = 10\%$ of fork and join connectors, e.g. for 1,000 components there are 10 fork and 10 join connectors. The cost of component communication is insignificant and thus not evident.



(a) Simple Composition Structure (n, τ)



(b) Complex Composition Structure (n, τ)

Fig. 7. Total Validation Cost

As expected, for a composed model with $n = 1,000$ and $\tau = 20$, validation time increases from 10 minutes for the simple composition in Fig. 7(a) to around 17 minutes for the composed model with a more complex structure in Fig. 7(b). Moreover, the percentage of validation cost also changes. In the composed model without fork and join connectors, the percentage of validation cost was distributed as $50 : 19 : 31$ for component coordination, component computation, and validation by comparison with reference model respectively. In contrast, the presence of fork and join connectors leads to a $40 : 15 : 45$ cost distribution, suggesting that as the number and complexity of connectors increase, the penalty incurred by the comparison with a reference model also increases. However, the cost of verifying component coordination becomes unfeasible as suggested before [12] and is capped as discussed in Section III-B. An important point to highlight is that the total execution cost for a SSF implementation [24] of the simple composed model with $n = 1,000$ is on average 49.57 seconds, and the cost of validating it is on average 11.28 minutes. However, because of the layered nature of the validation approach, the cost incrementally increases and the user can stop at any validation layer as desired. The validation of component communication and coordination incurs an initial overhead of 5.5 minutes. The meta-simulation layer incurs and additional 2.14 minutes. Finally, the comparison with a reference model incurs a cost of 3.64 minutes. However, space constraints prevent us from showing the analysis here.

IV. Related Work

Verification, Validation, and Accreditation (VV&A) has been a principal focus of research in the simulation community since the late 70s [25], with work such as that by Balci [3], [26] and Sargent [26], [16], [25] among others, providing detailed guidelines and process organization. However, very few works quantify the cost of validation, more so in the new area of component-based modeling and simulation. Balci and Sargent propose a methodology for cost-risk analysis in the statistical validation of simulation models, by looking at the relationship between data collection cost, acceptable validity range, and model builder and model user risks, when performing statistical hypothesis testing [26]. A validity measure is proposed and the trade-offs between data collection budget and model user's risk are analyzed. However, there is no estimate of the actual cost of validation. Historical data from the US Defense Modeling and Simulation Office suggests that VV&A activities account for 5 to 17.5% of the total modeling and simulation budget [4]. A clear estimate of the validation cost and the key factors that influence it is also missing in the simulation industry. Reports from industry suggest that validation cost is between 5 and 19% of the total project cost [27] but no cost model is provided. The same applies to the validation of component-based simulations. While there are many works that focus on verifying and validating general model properties [21], [9], [10] or model execution [2], [11], few present a break-down of the validation cost. In contrast, we evaluate four recent approaches based on two main perspectives, the validation of model properties and the validation of model execution. We study the cost of validation based on the number

of components, the validation window size, and the decree of component connectivity between components.

V. Conclusion

Industry practice suggests that the cost of traditional simulation validation ranges from 5 to 19% of the entire project cost [27]. In component-based modeling and simulation, semantic validation cost can be significantly higher. To the best of our knowledge, we present the first quantitative study of the cost of validating semantic composability and its trade-offs with validation accuracy. This study compares the cost of four main validation approaches, namely CD++ DEVS [10], the Z specification based DEVS [9], the Petty & Weisel formal theory [2], and deny-validity [11]. A simple queueing network model with 1,000 components is used and validation measurement is performed on a Dell PowerEdge SC1430 compute server. The key factors that influence the computational cost of validation are the characteristics of the *simulation problem* such as the composition size and the degrees of component interaction, and characteristics of the *validation approach* such as the techniques used and the levels of abstraction. Current validation approaches focus on validating general model properties, and on comparing between the composed model and a reference model. In general, the cost of validating model properties grows, as expected, with the number of components. However, the actual cost is vastly different among approaches, ranging from less than one second in CD++ DEVS to more than *seven minutes* in deny-validity.

In validating model execution, the time-based deny-validity approach costs seven times more than the Petty and Weisel timeless formalism. Our study showed that the cost of time-based ordering in the deny-validity approach explodes with increasing validation window size. Specifically, a 25% increase in validation window size results in a five-fold increase in validation cost. In addition, time-based ordering in the deny-validity approach, which facilitates the validation of composed models with more complex structures such as feedback loops and fork and join connectors, causes a significant increase in the validation cost.

As deny-validity is one of the most comprehensive approaches for semantic validation, we analyze it further. Firstly, for the same example, the total validation time is on average 17 minutes as compared to the simulation execution elapsed time of 49 seconds. Secondly, validation of general model properties accounts for 55 % of total validation time, with the remaining 45% incurred by model execution validation. Lastly, component interaction also increases validation cost, and a 10% increase in fork and join connectors increases validation cost by 50%.

References

[1] S. Kasputis and H. Ng, "Composable Simulations," in *Proc. of the Winter Simulation Conference*, USA, 2000, pp. 1577–1584.

[2] M. Petty, E. Weisel, and R. Mielke, "Composability Theory Overview and Update," in *Spring Interoperability Workshop*, USA, 2005.

[3] O. Balci, "Verification, Validation and Accreditation of Simulation Models," in *Proc. of the Winter Simulation Conference*, USA, 1997, pp. 135–141.

[4] Defense Modeling and Simulation Office (DMSO), *Verification, Validation, and Accreditation*. U.S. Dept. of Defense, 1996.

[5] A. Tolk, "What Comes After the Semantic Web - PADS Implications for the Dynamic Web," in *Proc. of the ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation*, Singapore, 2006, pp. 55–62.

[6] R. Bartholet, D. Brogan, P. Reynolds, and J. Carnahan, "In Search of the Philosopher's Stone: Simulation Composability Versus Component-Based Software Design," in *Proc. of the Fall Interoperability Workshop*, USA, 2004.

[7] R. Gore and P. Reynolds, "Applying Causal Inference to Understand Emergent Behavior," in *Proc. of the Winter Simulation Conference*, USA, 2008, pp. 712–721.

[8] P. Gustavson and L. Root, "Object Model Use Cases: A Mechanism for Capturing Requirements and Supporting BOM Reuse," in *Spring Simulation Interoperability Workshop*, Orlando, USA, 1999.

[9] M. K. Traore, "Analyzing Static and Temporal Properties of Simulation Models," in *Proc. of the Winter Simulation Conference*, Monterey, USA, 2006, pp. 897–904.

[10] G. Wainer, L. Morihama, and V. Passuello, "Automatic Verification of DEVS Models," in *Spring Interoperability Workshop*, USA, 2002.

[11] C. Szabo and Y. M. Teo, "An Approach for Validation of Semantic Composability in Simulation Models," in *Proc. of the ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation*, USA, 2009, pp. 3–10.

[12] K. E. Kennedy, "Formal Methods in the Verification and Validation of Simulation Models," in *Proc. of the Spring Interoperability Workshop*.

[13] D. K. Pace, "Modeling and Simulation Verification and Validation Challenges," *Johns Hopkins APL Technical Digest*, vol. 25, pp. 163–172, 2004.

[14] Y. M. Teo and C. Szabo, "CODES: An Integrated Approach to Composable Modeling and Simulation," in *Proc. of the Annual Simulation Symposium*, Canada, 2008, pp. 103–110.

[15] O. Dalle, "OSA: An Open Component-based Architecture for Discrete-event Simulation," in *Proc. of the European Conference on Modeling and Simulation*, Czech Republic, 2006.

[16] R. Sargent, "Verification, Validation, and Accreditation of Simulation Models," in *Proc. of the Winter Simulation Conference*, USA, 2000, pp. 50–59.

[17] J. Srba, "On the Power of Labels in Transition Systems," in *Proc. of International Conference on Concurrency Theory*, Denmark, 2001, pp. 277–291.

[18] G. Smith, F. Kammller, and T. Santen, "Encoding Object-Z in Isabelle/HOL," in *Proc. of the Intlernational Conference of B and Z Users*, France, 2002, pp. 82–89.

[19] M. Petty and E. Weisel, "Basis for a Theory of Semantic Composability," in *Proc. of the Spring Interoperability Workshop*, USA, 2003.

[20] A. Tolk, S. Y. Diallo, and C. D. Turnitsa, "Mathematical Models Towards Self-organizing Formal Federation Languages Based on Conceptual Models of Information Exchange Capabilities," in *Proc. of the Winter Simulation Conference*, USA, 2008, pp. 966–974.

[21] F. Moradi, R. Ayani, S. Mokarizadeh, G. H. A. Shahmirzadi, and G. Tan, "A Rule-based Approach to Syntactic and Semantic Composition of BOMs," in *Proc. of the IEEE Symposium on Distributed Simulation and Real-Time Applications*, Greece, 2007, pp. 145–155.

[22] M. Ben-Ari, *Principles of the Spin Model Checker*. Springer Verlag, 2008.

[23] H. Garavel, F. Lang, R. Mateescu, and W. Serwe, "CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes," in *Proc. of the International Conference on Computer Aided Verification*, Berlin, Germany, 2007, pp. 158–163.

[24] J. Cowie, "Towards Realistic Million-Node Internet Simulations," in *Proc. of International Conference on Parallel and Distributed Processing Techniques and Applications*, USA, 1999, pp. 2129–2135.

[25] R. G. Sargent, "Validation of Simulation Models," in *Proc. of the Winter Simulation Conference*, USA, 1979, pp. 497–503.

[26] O. Balci and R. G. Sargent, "A Methodology for Cost-Risk Analysis in the Statistical Validation of Simulation Models," *Communications of the ACM*, vol. 24, pp. 190–197, 1981.

[27] G. Love and G. Back, "Model Verification and Validation for Rapidly Developed Simulation Models: Balancing Cost and Theory," in *Proc. of the Conference of the System Dynamics Society*, Belgium, 2000.

Fairness Verification of BOM-based composed models using Petri Nets

Imran Mahmood, Rassul Ayani, Vladimir Vlassov
KTH Royal Institute of Technology
Stockholm, Sweden
{imahmood, ayani, vladv}@kth.se

Farshad Moradi
Swedish Defense Research Agency (FOI)
Stockholm, Sweden
farshad@foi.se

***Abstract*—Model reuse is a promising and appealing convention for effective development of simulation systems. However it poses daunting challenges to various issues in research such as *Reusability* and *Composability* in model integration. Various methodological advances in this area have given rise to the development of different component reusability frameworks such as BOM (Base Object Model). However, lack of component matching and support for composability verification and validation makes it difficult to achieve effective and meaningful reuse. For this reason there is a need for adequate methods to verify and validate composability of a BOM based composed model. A verified composed model ensures the satisfaction of desired system properties. Fairness, as defined in section II, is an important system property which ensures that no component in a composition is delayed indefinitely. Fairness in a composed model guarantees the participation of all components in order to achieve the desired objectives.**

In this paper we focus on verification and propose to transform a composed BOM into a Petri Nets model and use different analysis techniques to perform its verification. We propose an algorithm to verify fairness property and provide a case study of a manufacturing system to explain our approach.

Keywords*—*Model Verification;Composability;BOM framework; Petri Nets Analysis; Fairness; Manufacturing System.

I. INTRODUCTION

In the last two decades, the defense industry has invested significant resources in technologies and methods for making independently developed simulations work together [1]. The defense industry gained substantial experiences in interconnecting various simulation systems and the simulation research community has developed some supportive theories under the rubric of simulation composability [1]. The main driving factors behind composability are to enable reuse of existing solutions, cost reductions and cross-domain solutions [2].

The recent discussion on concepts of composability was reignited by Petty's and Weisel's publications on theory of composability [3] according to which, "Composability is the capability to select and assemble simulation components in various combinations into simulation systems to satisfy specific user requirements". Component-based software engineering has been identified as a key enabler in the construction of composable simulations [4]. At the level of an abstract model, composability is the creation of a complex model from a collection of basic reusable model components [5]. Composability is an effective way to achieve reusability therefore at the model level reuse relies on a composition framework that provides features for both composability and the mapping of composite models into executable form.

Base Object Model (BOM) is a component architecture based on these specifications. It contributes to conceptual modeling by providing the needed formalism and influence the ability to develop and compose model components [6] [7]. BOM is a Simulation Interoperability Standards Organization (SISO) standard. BOM encapsulates information needed to describe a simulation model using XML notation. BOM was introduced as a conceptual modeling framework for HLA (High Level Architecture) which is an IEEE standard for distributed simulations. In BOM different elements such as entities, events, actions and state-machines of the components are defined. Entities, events and actions represent the structural information about the real world objects that are being modeled, whereas State-machine is an essential part of BOM that provides means to formalize the component behavior and is our focus in this paper. Without the loss of generality we assume in this paper that a BOM component represents one entity. For details interested readers should refer to [8] [9].

The BOM framework poses an adequate potential for effective model composability and reuse; however it lacks means to express necessary elements of semantic accordance (agreement on the understanding of mutual communication) and behavioral coherency (having an interaction consistent with the common goals) between the composed components, which are essential for reasoning about the validity of the composition [10]. This fact leads us to the investigation of external methods for the matching and verification of BOM composability.

In modeling and simulation, verification is typically defined as the process of determining whether the model has been implemented correctly [11]. Actually, verification is concerned with the accuracy of transforming the model's requirements into a conceptual model and the conceptual model into an executable model [12]. We focus on the former part and assume that the behavioral correctness is a part of the model's requirements. In our case the conceptual model is represented by a composed BOM. Our task is to verify that the BOM based composed model satisfies the behavioral requirements such as avoiding deadlock and live-lock or guaranteeing fairness. These requirements are defined in form of system properties and may also include specific reachability properties representing certain desirable or undesirable incidences in the system. All these properties can generally be grouped as *Safety* or *Liveness* requirements. In *Composability Verification* we assess that the model components are correctly assembled such that they satisfy the given requirement specification and their combined behavior is suitable to reach given objectives.

Fairness property as defined in the next section, has a significant place in requirement specifications, and the motivation behind the notion of verifying fairness in a composed model is to disallow infinite executions of some components due to which others are unable to proceed or make progress [13]. It is possible that a deadlock-free composed model makes progress but it cannot guarantee the fulfillment of desired objectives because one of the

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

components, whose participation is essential, may not get a chance to involve in the interaction. However if a composed model satisfies fairness property, we can guarantee that all components will get a chance to interact and thus their combined behavior influences in a positive way the ability to reach the desired objectives.

In order to achieve suitable composability, the correctness of a composed model is evaluated in various levels of consistencies namely *syntactic*, *semantics* and *pragmatics* [14]. *Syntactic consistency* means that two models can fit together, i.e., the output of one can be read as an input to the other, whereas *Semantic consistency* is concerned with the meaningful interaction of the components. It is further divided into *Static-Semantic* and *Dynamic-Semantic* sublevels. The former involves in having a concise and mutual understanding of the data exchanged by the components participating in the composition, while the latter deals with having a conforming behavior towards their collective objectives [10]. *Pragmatic consistency* on the other hand refers to a context based meaningful interaction of the models [14].

Based on different consistencies involved in the process of model composition and the fact that the BOM framework lacks built-in verification techniques to evaluate them, different approaches have been suggested for the external composability verification. A rule-based approach proposes to match a set of BOMs and verify their syntactic, static-semantic and dynamic-semantic consistencies [10]. Another approach covers different aspects of validation of semantic consistency in composability of components [15]. A similar work [16] suggests an instrumentation technique to specify verification criteria as a *Model Tester* and check dynamic semantic composability. Using the model tester, various system properties can be modeled as a state-machine component and attached to the composition like a device tester. Another method [17] explores the possibility of using Petri Nets (PN) formalism for the dynamic-semantic composability verification. In this approach a BOM based composed model is transformed into a single PN model and PN Reachability analysis are applied to verify the model behavior [17]. In another similar work presented in [18], an automatic conversion of an architecture created using UML to a Colored Petri Nets based executable model is suggested for determining the potential behavior and performance of the system.

In this paper, we revisit the verification of BOM composability using Petri Nets formalism [17], and utilize certain PN properties analysis methods to verify fairness in the composed model. Here we only consider classical Petri Nets leaving consideration of Timed, Hierarchical and Colored PN extensions to our future work. We present theorems and PN properties required in the fairness verification, and provide a fairness verification algorithm. We also include a case study to explain our approach. The approach to fairness verification presented in this paper can be generalized and applied to verify various other system properties in the given model. It should be noted that the focus of this paper is not to introduce new techniques in Petri Nets; instead we suggest the utilization of existing techniques from Petri Nets community for the purpose of Composability verification. We however propose minor refinements to suit our needs. In our observation, using PN for verification proves to be more fruitful as compared to other similar formalisms, due to the broader range of verification methods and tools and techniques and due to their greater suitability in formal representation of concurrent behaviors.

The rest of the paper is organized as follows: Section II briefly provides basic concepts and definitions of Petri Net formalism that are used in this paper. Section III formulates our Verification approach and presents our algorithm for fairness verification. Section IV provides details on the implementation of our verification process and framework. Section V covers a manufacturing system case study and its counterexample; whereas section VI concludes the paper.

II. PRELIMINARILY CONCEPTS AND NOTATIONS

In this section we briefly discuss basic concepts of Petri Nets theory used in our work. Readers should refer to [18], [19] and [20] for details. We also define and discuss the essentials of fairness property in this section.

A. Petri Nets

Petri Nets (PN) is a graphical and mathematical tool for the formal description of the flow of activities in complex systems. It is particularly suited to represent typical situations like synchronization, sequentiality (producer-consumer problem), concurrency and conflict (mutual exclusion) in a system.

Mathematically a PN is a 5 tuple: $\mathbf{PN = \langle P, T, F, W, M_0 \rangle}$ where:

- P is a finite set of places $P = \{ p_1, p_2, \ldots, p_n \}$
- T is a finite set of transitions $T = \{ t_1, t_2, \ldots, t_n \}$
- F is a set of arcs $F \subseteq (P \times T) \cup (T \times P) \mid P \cap T = \phi$ and $P \cup T \neq \phi$
  (In words: an arc can be connected from place to transition or vice versa but not from place to place or transition to transition).
- W: $F \rightarrow \{1, 2, 3\ldots\}$ is an arc weight function.
- M_0 is the initial marking.

Graphically, a PN is a bipartite graph that has two types of nodes: Circular (or oval) nodes represent places whereas box (or rectangular) nodes represent transitions. These nodes are connected through arcs. A dot in a place represents token. Following are some selected terms and definitions from PN literature that we use in this paper. (It is assumed throughout the paper that a PN has ***m-places*** and ***n-transitions***).

1) Marking

A marking $\mathbf{M: P \rightarrow N}$ is the state of a PN that represents the number of tokens in each place. **M** is a non-negative $\mathbf{m \times 1}$ integer-valued vector $\mathbf{M \in N^m}$ where i^{th} component $\mathbf{M_i}$ represents the token load $\mathbf{M(p_i)}$ of the i^{th} place.

2) Arc Weight

Arc weight **w(p,t)** denotes the number of required tokens to be consumed from the input places whereas **w(t,p)** denotes the number of tokens produced to the output place.

3) Enabled Transistion

For a given marking **M** a transition $\mathbf{t \in T}$ is said to be enabled if $\mathbf{\forall p \in {}^\bullet t,\ m(p) \geq w(p,t)}$ i.e, all the input places of transistion **t** have number of tokens greater or equal to the input arc weight.

(Note that $\forall p \in {}^\bullet t$ is the dot notation, which represents all input places that are connected to the transition t. Similarly

$\forall p \in t^{\bullet}$ represents all the output places that are connected to transition t)

4) Firing Count Vector

An **n×1** column vector **X** of nonnegative integers is called firing count vector, where the i^{th} entry of **X** denotes the number of times transistion **t** must be fired to transform $\mathbf{M_0}$ to **M**.

B. Matrix Definitional Form (MDF)

The Matrix Definitional Form (MDF) of a PN consists of the matrices: $\mathbf{A^+}$, $\mathbf{A^-}$, **A**

- $\mathbf{A^+ = [a^+_{ij}]_{nxm}}$ is the *output matrix*, where $\mathbf{a^+_{ij} = w(t_i, p_j)}$; if $\mathbf{p_j \in t_i^{\bullet}}$, and $\mathbf{i \in n}$; $\mathbf{j \in m}$ i.e., if $\mathbf{p_j}$ is connected to the output of $\mathbf{t_i}$ then $\mathbf{a^+_{ij}}$ is equal to the weight of output arc; 0 otherwise.
- $\mathbf{A^- = [a^-_{ij}]_{nxm}}$ is the *input matrix*, where $\mathbf{a^-_{ij} = w(p_j, t_i)}$; if $\mathbf{p_j \in {}^{\bullet}t_i}$, i.e., if $\mathbf{p_j}$ is connected as input to $\mathbf{t_i}$ then $\mathbf{a^-_{ij}}$ is equal to the weight of input arc; 0 otherwise.
- $\mathbf{A = [A^+ - A^-]_{nxm}}$ is the *incidence matrix*. The i^{th} row in **A** denotes the change of the marking as a result of firing transition $\mathbf{t_i}$.

C. State Equation

The incidence matrix **A** is used to compute any reachable marking **M** using the following State Equation:

$$\mathbf{M = M_0 + A.X}$$

Where:

- $\mathbf{M_0}$ is the initial Marking
- **A** is incidence matrix
- **X** is the firing count vector

D. Bounded Petri Nets

Boundedness in Petri Nets is a safety property. A place is bounded with k, if the token count does not exceed k in any marking of a PN. A PN is k-bounded if each place is k-bounded.

E. Invariants

Occurrences of transitions transform the token distribution of a net, but often respect some global properties of markings, regarded as *Linear Invariant Laws*. Invariants are useful for analyzing structural and behavioral properties of Petri Nets. Two most important invariants are the following.

a) P-Invariants

Place Invariants formalize invariant properties regarding places in PN, e.g., if in a set of places the sum of tokens remains unchanged after firing, then this set can define a place invariant. They are useful to evaluate structural properties of PN. P-Invariant of a PN can be formed if there exists an **m×1** vector **Y**, such that for any reachable marking **M**

$$\mathbf{M.Y = M_0.Y \text{ and } A.Y = 0}$$

If there exist a P-Invariant such that $\mathbf{Y(p)>0}$, for all $\mathbf{p \in P}$, then PN is guaranteed to be structurally bounded [20].

b) T-Invariants

Transition Invariants on the other hand formalize properties regarding transition firing sequences applicable to a PN. They are useful to evaluate behavioral properties such as liveliness and fairness. A firing count vector X, is called a T-Invariant if: $\mathbf{A^T.X \geq 0}$

I.e., firing each transition the number of times specified in X, brings the PN back to its initial marking M_0 [20]. There can be multiple T-invariants for a PN, though a minimal T-Invariant is called the *Reproduction vector* of the net [21].

F. Fairness Property

Informally a system is said to be fair if: "*No component of the system that becomes enabled sufficiently often should be delayed indefinitely*" [13]. On the basis of the extent of sufficiency, fairness is generally categorized in the following three types in literature:

a) Unconditional (or Impartial) Fairness

Every component in a system proceeds infinitely often. (Unconditionally)

b) Weak (or Just) fairness

Every component in a system that is enabled continuously from some point onwards eventually proceeds.

c) Strong fairness

Every component in a system that is enabled infinitely often proceeds infinitely often.

Unconditional fairness is also known as non-deterministic choice and is usually present among the components that are independent of each other. A noticeable difference in weak and strong fairness is that weak fairness involves persistent enabling of a component that wants to proceed, whereas strong fairness is not persistently enabled [13].

Some important generalizations of fairness exist in literature:

a) Equi-fairness: To give each component an equal chance to proceed. This type of fairness does not always apply in real world scenarios because of priority policies or some other reasons.

b) Bounded fairness: To give each component an equal number of chances, such that no component proceeds for more then "k-times" without letting the others to take their turn.

In Petri Nets, fairness can be viewed in two perspectives namely: *Transition fairness* and *Marking fairness.* The former corresponds to fairness of choice of transitions, and the latter deals with the fair reachability of states [13]. In this paper we focus on Transition fairness.

III. VERIFICATION APPROACH

In this section, we discuss our approach for the verification of a BOM based composed model that concentrates on the *"Dynamic-Semantic"* consistency and tests that the composition posses correct behavior that satisfies given specifications and objectives. In this paper, we consider "Fairness" as a specification criterion for composability verification. We divide our approach in two main phases: (1) Transformation, (2) PN Analysis, explained below.

A. Transformation

BOM framework uses XML notation for representation. An essential part of BOM is State-machine, which provides means to model the abstract behavior of each component

participating in the composition and is our main concern in the composability verification.

In this phase we transform the state-machines of all members of the composed BOM in to a single PN model. This is an n→1 automatic transformation. In this transformation, we consider each state (of state-machines) as Place in PN and each event (send or receive event) as a Transition in PN such that the sender state **s** and the receiver state **r** (of two different state-machines in BOM) have two places **p** and **q** as input places in PN and connect to a single transition **t** in PN representing the event they exchange. The next states **s´** and r´(after sending or receiving event) have two places **p´** and **q´** in PN as output places such that they have output arcs coming from the transition **t**. We assume that the instances of each component are represented by the tokens (in the initial marking of PN) assigned to the places corresponding to the initial states in the state-machines. The number of tokens (instances) is given as an input parameter.

The transformation process is complete, when all the states and events of every state-machine in BOM are plotted in the PN model such that no element is duplicated, and each place or transition is connected so that there are no broken links. Figure 1 depicts an example of the transformation.

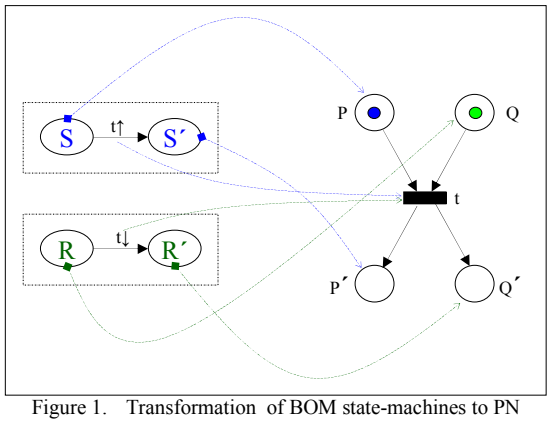


Figure 1. Transformation of BOM state-machines to PN

B. PN Analysis

System properties are explored in this phase depending on the requirements specifications. In Petri Nets, system properties are divided into two groups, namely, structural properties and behavioral properties. Structural properties focus on the structure of the net and are independent of the initial marking; whereas behavioral properties concentrate on the model behavior and depend on the initial marking [18]. Some of the important behavioral properties are *Reachability, Boundedness, Liveness (Deadlock freeness, Livelock freeness)* and *Fairness*. Depending on the nature of the property, various methods have been proposed for verification. These methods include *Algebraic techniques*, *Reachability graph analysis*, *Model Checking* etc. For example, in [17] we have already presented an approach to deadlock detection in a composed model using Reachability graph analysis. In this approach we transformed a composed BOM into PN and explored the Reachability graph for any dead marking. If there exist a dead marking from where no further transition is possible then a deadlock is detected [17].

In this section, we discuss the technique for the verification of fairness property and provide the necessary and sufficient conditions for a PN model to be fair. The evaluation of these conditions in a PN model involves linear algebraic computations; therefore we classify this approach as an *Algebraic technique*. Based on the theorems below, we propose an algorithm for automatic fairness verification.

In Petri Nets, fairness is mainly perceived in terms of occurrences (or proceedings) of transitions. Two transitions $\mathbf{t_1}$ and $\mathbf{t_2}$ are said to be in a fair relation if there exists a positive integer **k** such that for any reachable marking **M** and any firing sequence **σ:**

$$\#(t_1/\sigma) = 0 \Rightarrow \#(t_2/\sigma) \leq k$$

and

$$\#(t_2/\sigma) = 0 \Rightarrow \#(t_1/\sigma) \leq k$$

Where #(t/**σ**) denotes the number of occurrences of transition t in a firing sequence **σ**. In words, neither of the transitions should occur more than a finite number of times (k) without letting the other to do so for at least once. This is known as *bounded fairness* (or *B-Fairness*). If every pair of transition is in a bounded fair relation, then the entire net is said to be fair [21]. For the algebraic verification of fairness in a PN model we rely on the following theorems. Details and proofs of these theorems are discussed in [21].

Theorem I:

Given a PN, if there exists a firing-count vector X, such that: $\mathbf{A^T.X \geq 0}$ and $\mathbf{X \neq 0}$ *then a necessary condition for the PN to be fair is that each entry of* ***X*** *is positive.*

Theorem II:

If a Petri Net ***N*** *is bounded for any initial marking* $\boldsymbol{M_0}$ *then the condition in Theorem I is necessary and sufficient for N to be fair.*

Corollary: If there exist a P-Invariant **Y** *with* $\mathbf{Y(p) > 0}$ *for all* $\mathbf{p \in P}$ *and* $\mathbf{A.Y=0}$*, then the PN is guaranteed to be structurally bounded.*

Theorem III:

A fair Petri Net PN has only one reproduction vector (i.e., a minimal T-Invariant) at the most.

Based on the above definition of fairness and theorems I, II and III, it can be inferred that if there exists a **single** T-Invariant X for a given PN model, i.e., a firing count vector where each entry is non-zero and $A^T.X \geq 0$, and if at least one P-Invariant exists, then we can say that the net is fair.

Based on the above discussion, we propose the fairness verification algorithm presented in Figure 2.

Algorithm 1: Fairness Verification

Input: M_0, A
Result: True/False

```
1  begin
2      ListX_T= Call GetTInvairants()
3      if |ListX_T|=1 and A.X_T≥ 0 and each x in X_T>0 then
4          ListY_P= Call GetPInvairants()
5                  if |ListY_P|> 0 then
7                          return true
8                  else
9                          return false
10                 end if
11     else
12         return false
13     end if
14 end
```

Figure 2. Fairness Verification algorithm

In the beginning of the procedure, a sub routine "GetT-Invariants" is called (2) that returns a list of all possible T-invariants of the given PN. If only one T-invariant exists (3) and the multiplication of the T-Invariant with the incidence matrix gives a non-negative result and each entry in the T-invariant $\mathbf{X_T}$ is non-zero then we say that conditions in Theorems I and III are fulfilled. Then we call GetP-Invariant (4). If a non-zero list is returned then it satisfies Theorem II providing that the given PN is structurally bounded thus the given PN is fair. If either or both of tests in (3) and (5) fail, we conclude that the net is not fair.

The methods for calculating P-Invariants (*GetPInvariant*) and T-Invariants (*GetTInvariant*) of a PN model have been extensively studied. The details of these procedures are outside the scope of this paper, however we briefly describe the basic principle to compute the fundamental set of P-invariants and T-Invariants. The principle of finding P-Invariance is presented in Figure 3.

The input of the procedure is the Incidence Matrix **A** and an Identity matrix **B** of size **m×n**. The output is a matrix **B´** whose rows are the fundamental set of P-Invariants. The same procedure is used to find T-invariants but the Incidence Matrix is transposed.

Algorithm 2: P-Invariance (Farkas Method)

Input: A, B /* For T-Invariance A^T*/
Result: B´
1 begin
2 B´ := A|B /* Augmentation of A with n×n identity matrix B */
3 **for** i=1 to n /* n = |T| */
4 **for** each pair of rows b_1, b_2 in B'_{i-1} where $b_1(i)$ and
5 $b_2(i)$ have opposite signs (i.e., annul the i^{th} column)
6 b := $|b_2(i)|.b_1 + |b_1(i)|.b_2$
7 b´ := b/gcd of each element of b
8 augment B'_{i-1} with b´
9 **end for**
10 Delete all rows of B'_{i-1} whose i^{th} component is
11 different from 0, the result is B´
12 **end For**
13 Delete the first N columns of B´
14 **return** B´ /*Matrix whose rows contains fundamental P-Invariants*/
15end

Figure 3. P-Invariants calculation algorithm (see [22] and [23] for details) (gcd = Greatest common divisor)

This is the basic principle of calculating Invariance proposed by Julius Farkas in 1902. This method was introduced in the context of PN by J.M. Toudic and later refined by H. Alaiwan and G. Memmi. In this paper we derive the method of Invariance calculations from the much optimized algorithm discussed in [23].

IV. VERIFICATION FRAMEWORK

In this section, we present a composability verification framework as our main contribution. Our proposed framework automates the verification of BOM based composed models and focuses on the dynamic semantic (behavioral) consistency.

Our Java based framework relies on PIPE library [24] as an underlying layer for basic PN operations such as graphical preview and simulation of the PN model (which is not required in the verification process, but it helps to view and study the model). It also utilizes the functions for general matrix operations, constructing A^+, A^- and A matrices, finding P-Invariants and T-invariants etc. We have utilized these functions to implement our suggested verification algorithm. Figure 4 represents our proposed verification framework.

Our framework performs the following steps to verify the BOM composition as marked in Figure 4.

A. Input

A composed BOM XML document is given as an input which is parsed and the components are fetched. Verification requirements (RS) are also provided by the modeler at this step. In this paper, we propose a method for checking only fairness; however similar methods for other properties can be added in the framework as add-on modules in order to provide a wide range of automated tests, allowing the modeler to choose tests according to the verification requirements.

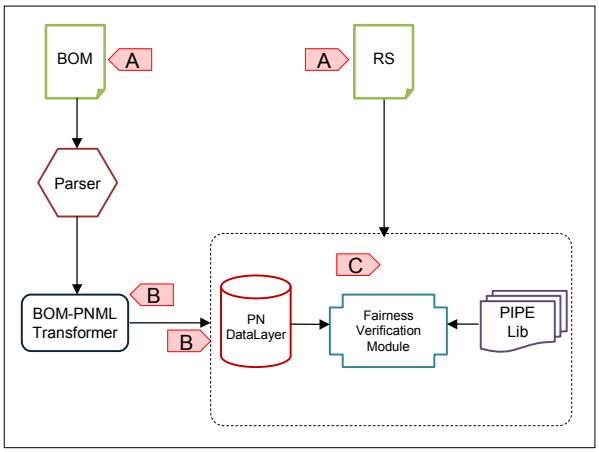


Figure 4. Composability verification framework

B. BOM to PNML Transformation

In this step we transform the composed BOM in to PNML (Petri Net Markup Language) format using our algorithm presented in [17]. PNML is a standard XML-based interchange format for Petri Nets and in our case it is used to represent the transformed composed model in a single place/transition PN model.

In this step, PNML code for Places, Transitions and Arcs is generated. Once the transformation is complete, a consistency check is performed to ensure that there are no broken paths. The output code is written to an XML file representing our composed model in PN format. This file is retaken as input in the framework as PN Data-Layer.

C. Fairness Verification

An automated routine initiates the step mentioned in Algorithm 1 (Figure 2). First, the PN Data layer is accessed and the PN model is populated in a DOM document object making it easy to navigate and access arbitrary data from the PN. Then, A^+, A^- and A matrices are constructed and the initial marking vector M_0 is initialized. Next, we calculate T-invariants of the given PN using functions from the PIPE library. We check that there is only one T-invariant $\mathbf{X_T}$ and all the entries in $\mathbf{X_T}$ are non-zero. Then we check whether the multiplication of A and $\mathbf{X_T}$ returns a null matrix. If all these conditions are fulfilled then we calculate P-invariants. If there exists at least one P-invariant then we are sure that the net is structurally bounded and that the given PN is fair.

V. Case Study

In this section, in order to explain our approach, we discuss a case study of the model of a manufacturing system. This model is composed of three BOM components namely; *Machine-A, Machine-B and a Robot.* Both machines share Robot as a resource that loads raw material on them. Either of the machines can take Robot one at a time. Once the Robot is acquired, it loads raw material on the machine. After the material has been loaded, the machine releases the Robot, proceeds with the production process, and, finally, outputs a finished product. Once the Robot is released, it can be taken by the other machine. Figure 5 shows the components and the workflow of manufacturing system.

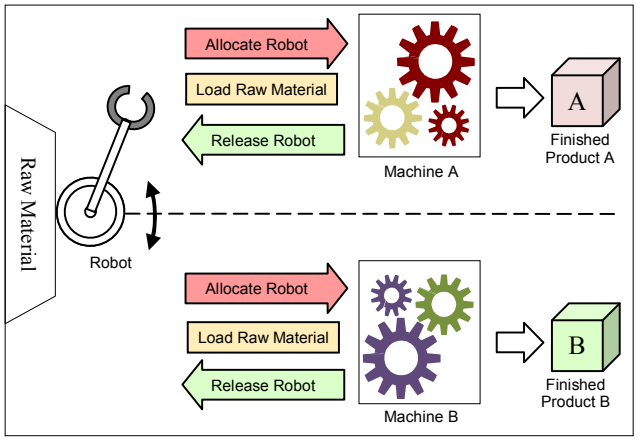


Figure 5. Manufacturing system

All components are built and composed using the BOM framework. We assume for simplicity that the send events are not lost and will eventually be received by the receivers. We also assume that a next send-event is issued only after the previous send-event has been consumed. Figures 6, 7 and 8 depict simplified state-machines of each component involved in the composition. Note that the robot (Figure 8) can send either of the send-events with a random choice.

Given the BOM components of the manufacturing system our objective is to verify that the composed model of the system is fair, i.e., neither of the machines takes the robot more than a certain number of times without letting other to take it at least once.

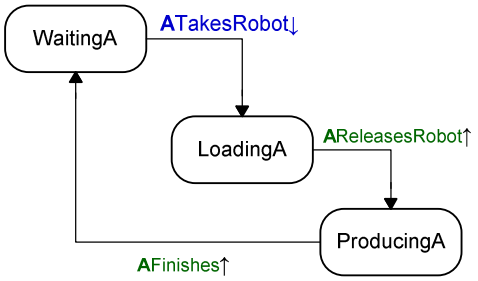


Figure 6. FSM of Machine A

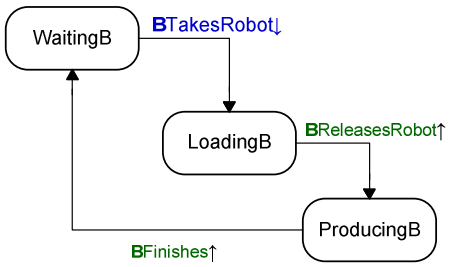


Figure 7. FSM of Machine B

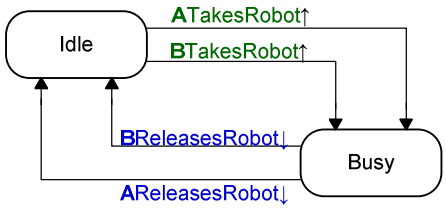


Figure 8. FSM of Robot
[↑=SendEvent] [↓=Receive Event]

The fairness verification is performed as follows.

A. Fairness Verification

In the first step, the BOM xml document is parsed and the state-machine components are fetched. Then, using the BOM-to-PNML transformation algorithm [17], a PNML file is generated. This file is parsed and a DOM object is initialized that acts as a Petri Net Data layer. The required number of tokens is initialized for each place during this transformation step. Figure 9 demonstrates the generated PN model.

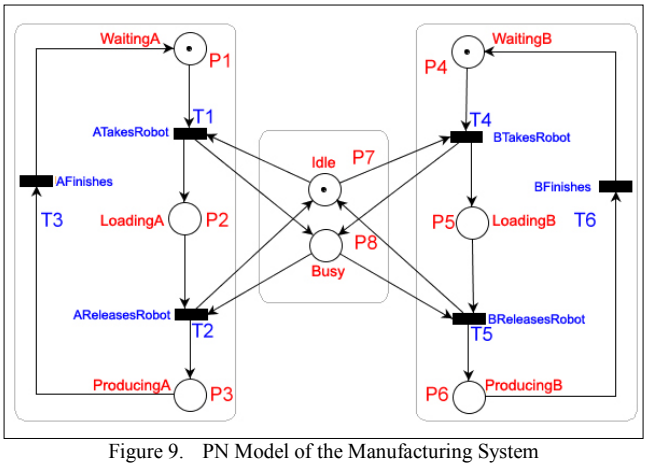


Figure 9. PN Model of the Manufacturing System

In the initialization phase, the initial marking $\mathbf{M_0}$ and the Incidence Matrix $\mathbf{A}$ are calculated as shown below.

$\mathbf{M_0}$	P1	P2	P3	P4	P5	P6	P7	P8
	1	0	0	1	0	0	1	0

A	P1	P2	P3	P4	P5	P6	P7	P8
T1	-1	1	0	0	0	0	-1	1
T2	0	-1	1	0	0	0	1	-1
T3	1	0	-1	0	0	0	0	0
T4	0	0	0	-1	1	0	-1	1
T5	0	0	0	0	-1	1	1	-1
T6	0	0	0	1	0	-1	0	0

Note that the labels of rows and columns in **A** and elements in $\mathbf{M_0}$ correspond to places and transitions in Figure 9. The matrix **A** is given as input to the Invariant calculation module that detects the following T-Invariants in the PN model of the Manufacturing System:

T1	1
T2	1
T3	1
T4	0
T5	0
T6	0

T1	0
T2	0
T3	0
T4	1
T5	1
T6	1

As there are zero entries in the T-Invariants, so the net is unfair even if there exists any P-Invariant. As the PN is unfair, we cannot guarantee that neither of the machines will over performs by acquiring robot all the time without letting the other to get the robot for at least once. Therefore either of the machines may face a situation in which it is unable to produce enough number of products to meet the required objectives; consequently the composed model may fail to satisfy given specifications.

B. Counterexample

In order to understand the fairness verification process, we provide a counterexample. In this example we introduce another component called *Controller* in the composition that can supervise fairness. The job of *Controller* is to enforce fairness in the system. Figure 10 shows the BOM state machine of *Controller*.

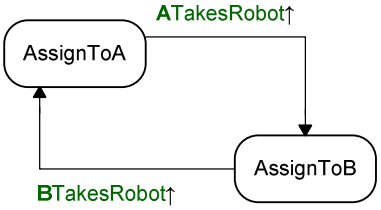


Figure 10. FSM of Controller

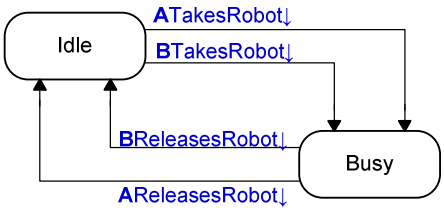


Figure 11. FSM of Modified Robot
Figure 12.

This component regulates the assignment of Robot to machines switching between them. We modify Robot to be a reactive component, which recieves ATakesRobot or BTakesRobot events from the controller, as shown in Figure 11. The PN model of the manufacturing system with added controller, obtained by BOM-to-PNML transformation, is shown in Figure 12.

If we look at the model from the concurrency point of view, we can notice that the *Controller* is acting as a semaphore, which allows both *Machine-A* and *Machine-B* to execute only one at a time.

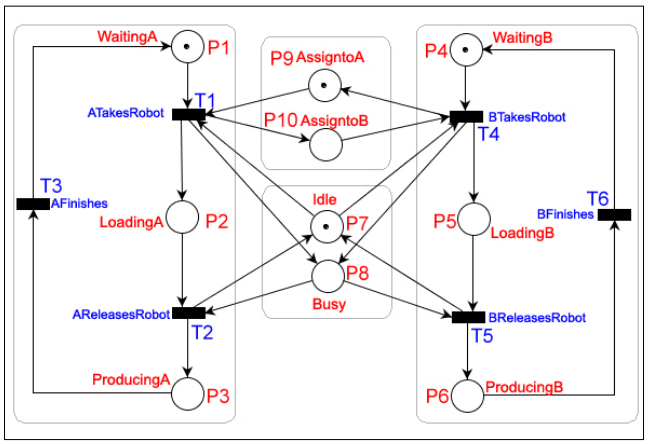


Figure 13. PN Model of the Manufacturing System with a controller

In order to verify fairness property we repeat our verification process for the PN model of counterexample. In the initialization phase, the initial marking $\mathbf{M_0}$ and Incidences Matrix **A** were calculated as follows.

M_0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
	1	0	0	1	0	0	1	0	1	0

A	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
T1	-1	1	0	0	0	0	-1	1	-1	1
T2	0	-1	1	0	0	0	1	-1	0	0
T3	1	0	-1	0	0	0	0	0	0	0
T4	0	0	0	-1	1	0	-1	1	1	-1
T5	0	0	0	0	-1	1	1	-1	0	0
T6	0	0	0	1	0	-1	0	0	0	0

After executing the verification process, we get the following T-Invariant and P-Invariant:

T1	1
T2	1
T3	1
T4	1
T5	1
T6	1

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
1	1	1	0	0	0	0	0	0	0

Having only one T-Invariant with non-zero entries and having a P-Invariant, satisfies the conditions required for the model to be *1-bounded fair*. Note that the existence of P-Invariant asserts that the net is structurally bounded. Also note that if we assign *k* tokens to the place P9 (Figure12) in the initial marking, the net will become *k-Bounded fair*.

Fairness property becomes significant in the composability verification of a composed model because it ensures the due participation of all components in order to achieve the given objectives. As illustrated by the case study of the manufacturing system, fairness of Robot allocation can ensure that both machines will perform to produce a required number of products. If there is no fairness we cannot guarantee that this objective will be reached.

VI. SUMMARY AND CONCLUSION

In this paper we discuss verification of BOM based composed models. We advocate that transforming a composed BOM into a Petri Nets model and applying existing Petri Nets analysis techniques is a very useful approach for accurate and

efficient verification. We also propose a verification framework that performs the automatic transformation of BOM to Petri nets and based on the requirement specifications executes the verification task. We suggest an approach for verifying fairness property in a PN model and provide an algorithm for it. Subsequently we provide a case study of a manufacturing process to explain and illustrate our approach. This framework can further be extended to accommodate methods to verify other system properties. This approach can be generalized and can be applied for verification of system properties in other component frameworks.

The usage of Petri Nets in the Composability analysis proves to be very useful and promising technique, especially with a focus on the dynamic behavior of the system, as Petri Nets is one of the competitive formalisms in concurrent behavioral representation. Furthermore, the analysis techniques contributed by the Petri Net community over a couple of decades provide a significant improvement on efficient and accurate reasoning regarding the model correctness.

We are further interested to explore methods for the verification of other properties, such as live-lock freeness, starvation freeness, reachability of desirable states. Currently we lack formalism for behavioral verification specification to specify requirements in a formal way. We intend to seek a suitable solution for it and extend our framework in this capacity.

REFERENCES

[1] E. H. Page, "Theory and Practice for Simulation Interconnection: Interoperability and Composability in Defense Simulation," in Handbook of Dynamic System Modeling.: Chapman & Hall, 2007, ch. 16.

[2] A. Tolk, "Interoperability and Composability," in Modeling and Simulation Fundamentals Theoretical Underpinnings and Practical Domains.: John Wiley, 2010, ch. 12.

[3] M. D. Petty and E. W. Weisel, "A theory of simulation composability," Virginia Modeling Analysis & Simulation Center, Old Dominion University, Norfolk, Virginia, 2004.

[4] R. G. Bartholet, D. C. Brogan, J. . P. F. Reynolds, and J. C. Carnahan, "In Search of the Philosopher's Stone: Simulation Composability Versus Component-Based Software Design," in Simulation Interoperability Workshop, Orlando, 2004.

[5] P. K. Davis and R. H. Anderson, Improving the composability of department of defense models and simulations.: RAND National Defense Research Institute, 2003.

[6] P. Gustavson and T. Chase, "Using XML and BOMS to rapidly compose simulations and simulation environments," in Winter Simulation Conference, Washington, DC, 2004.

[7] P. Gustavson and T. Chase, "Building Composable bridges between the conceptual space and the implementation space," in Proceedings of the winter simulation conference, Washington, DC, USA, 2007.

[8] P. Gustavson, "Guide for Base Object Model. Use and Implemenration," Simulation Interoperability Standard Organization (SISO), 2006.

[9] SISO's Base Object Model (BOM) Specification. http://www.boms.info/standards.htm (Last visited Dec, 2010)

[10] F. Moradi, R. Ayani, S. Mokarizadeh, G. H. A. Shahmirzadi, and G. Tan, "A Rule-based Approach to Syntactic and Semantic Composition of BOMs," in 11th IEEE Symposium on Distributed Simulation and Real-Time Applications, Chania, 2007.

[11] O. Balci, "Verification, Validation and Accreditation of simulation models," in Proceedings of the Winter Simulation Conference, Atlanta, GA, 1997.

[12] M. D. Petty, "Verification and Validation," in Principles of Modeling and Simulation.: John Wiley & Sons, 2009, ch. 6.

[13] M. Kwiatkowska, "Survey of fairness notions," Information and Software Technology, vol. 31, no. 7, September 1989.

[14] P. Davis, "Composability," in Defense Modeling, Simulation, and Analysis: Meeting the Challenge. Washington, D.C.: The National Academies Press, 2006.

[15] C. Szabo and Y. M. Teo, "An Approach for Validation of Semantic Composability in Simulation Models," in Principles of Advanced and Distributed Simulation, 2009. PADS '09, New York, 2009.

[16] I. Mahmood, R. Ayani, V. Vlassov, and F. Moradi, "Behavioral Verification of BOM based composed models," in 22nd European Modeling & Simulation Symposium, Fes, Morocco, Oct, 2010.

[17] I. Mahmood, R. Ayani, V. Vlassov, and F. Moradi, "Composability Test of BOM based models using Petri Nets," in 22nd IFIP International Conference on Testing Software and Systems, Natal, Brazil, Nov, 2010.

[18] L. W. Wagenhals, S. Haider, and A. H. Levis, "Synthesizing Executable Models of Object Oriented Architectures," in Proceedings of the conference on Application and theory of petri nets: formal methods in software engineering and defence systems, Adelaide, Australia, 2002.

[19] T. Murata, "Petri nets: Properties, analysis and applications 77(4), 541–580 (1989)," in Proceedings of the IEEE, 1989.

[20] M. Silva, Introducing petri nets.: Chapman and Hall, 1993.

[21] H.-C. Yen, "Introduction to Petri Net Theory," in Recent Advances in Formal Languages and Applications.: Springer Berlin, 2006, ch. 25.

[22] T. Murata and Z. Wu, "Fair relation and modified synchronic distances in a petri net," Journal of the Franklin Institute, vol. 320, no. 2, August 1985.

[23] J. Farkas, "Theory of simple inequalities," Journal of Pure and Applied Mathematics , vol. 1902, no. 124, 1902.

[24] M. D'Anna, "Concurrent system analysis using Petri nets: an optimized algorithm for finding net invariants," Computer Communications, vol. 11, no. 4, August 1988.

[25] J. Bloom et al., "Platform independent petri net editor," Imperial College, London, 2007.

Simulation of complex manufacturing systems via HLA-based infrastructure

Giulia Pedrielli, Paola Scavardone, Tullio Tolio
Mechanical Department
Politecnico di Milano
Milan, 20156, ITALY
giulia.pedrielli@mail.polimi.it

Marco Sacco, Walter Terkaj
Institute of Industrial Technologies and Automation (ITIA)
National Research Council (CNR)
Via Bassini 15
Milan, 20133, ITALY

Abstract— **Manufacturing systems can be thought as production networks nodes whose relations have a strong impact on design and analysis of each system. One of the most common techniques to support these tasks is Discrete-Event Simulation. The state-of-the art commercial simulators are already adopted to analyze complex networked systems, but the development of a monolithic simulation model can be too complex or even infeasible when a detailed description of the nodes is not available outside the "owner" of the node. In these cases the problem can be decomposed by modeling complex systems with various simulators that interoperate in a synchronized manner. Herein, the integration of simulators is addressed by taking as a reference the High Level Architecture (HLA) and the research carried out by Commercial-off-the-shelf Simulation Package Interoperability (CSPI) Product Development Group (PDG). This paper proposes modifications to CSPI-PDG protocols and to use patterns of how HLA can be effectively adopted to support CSP interoperability: a new solution for the synchronous entity passing problem and a modification to the Entity Transfer Specification are presented. The resulting infrastructure is validated and tested over a realistic industrial case.**

Complex Manufacturing Systems; Discrete Event Simualtion; Distributed Simulation; HLA

I. INTRODUCTION

Nowadays manufacturing companies have to face an increasingly turbulent market characterized by demand variability, unstable requirements from the clients and short product lifecycles [14]. In addition to the problems related to the market, the performance of a manufacturing system is deeply affected by its relations with other systems in the supply chain network. Indeed, every production system is not a standalone unit, but a node in a production network characterized by complex dynamics that should be considered during the design and analysis phases [18]. Both in the literature and in the industrial practice, Discrete Event Simulation is used to analyze production and logistics problems in various industrial domains [5] thanks to Commercial-Off-The-Shelf (COTS) Simulation Packages (CSPs) that are available on the market and provide a wide range of functionalities (e.g. visual building of the simulation model, simulation run support, animation, etc.). However, the complexity of a simulation model becomes hardly manageable when the relations between the nodes of a production network must be considered, since the modeling of a single node is already complex by itself and requires specific expertise and information [15]. Moreover, in real practice the developer of a simulation model can hardly access to all the information characterizing a production network because information sharing is seen as a threat by most of the companies, thus hindering the feasibility of developing a unique and monolithic simulation model to evaluate the performance of complex production networks. In these cases a distributed simulation approach can be proposed to face the aforementioned criticalities. However, even though a standard like HLA [4] has already been proposed to support the interoperability between simulators, several problems arise when trying to interoperate heterogeneous simulators in real industrial cases, thus highlighting the need of enhancing HLA with additional complementary standards [12]. Furthermore, the definition of a standard language for CSPs represents a relevant and not yet solved scientific challenge [3, 9, 10]. Recently, the COTS Simulation Package Interoperability-Product Development Group (CSPI-PDG), within the Simulation Interoperability Standards Organization (SISO), worked on the definition of the CSP interoperability problem (Interoperability Reference Models, IRMs) and on a draft proposal for a standard to support the CSPs interoperability (Entity Transfer Specification, ETS). Nevertheless, an effective interoperability among CSPs is still far to be reached in industrial contexts.

Boer [1] investigated the main benefits and criticalities related to the application of HLA in the industrial domain by carrying out a survey aimed at various players involved in the problem (e.g. simulation model developers, software houses, HLA experts). The results of the survey showed that CSPs vendors do not see direct benefits in using distributed simulation, whereas in industry HLA is considered troublesome because of the lack of experienced users and the complexity of the standard. A literature review on HLA applications in the civil domain was performed by the authors of this paper. About 100 papers published during the last decade were analyzed, showing that 30% of the works deal with the industrial domain. However, only 6% of the papers address industrial cases. The latter remark confirms the results obtained by Boer. This analysis highlights the need of further efforts to demonstrate the applicability of HLA and evaluate the benefits of interoperability in the industrial field.

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

The remainder of this paper is structured as follows. Section II presents the Problem Statement, delving into the problem of CSP interoperability and the IRMs. Section III analyzes the literature and highlights some of the open issues. Section IV proposes a solution to the Type A.2 IRM, a modification to ETS, and a communication protocol between the CSP and its adapter taking as a reference the work already carried out by CSPI-PDG. Section V addresses the implementation of the proposed solution that is validated (Section VI) and then tested over a realistic industrial case (Section VII). Finally, conclusions and future developments are drawn in Section VIII.

II. PROBLEM STATEMENT

The analysis of networked manufacturing systems by means of distributed discrete event simulation is addressed by developing distinct simulators for each subsystem, and connecting them according to the relations characterizing the network. The work presented in this paper is aimed at supporting the design and analysis tasks (Section I) for the following classes of manufacturing systems [2]:

- Assembly/disassembly systems, where assembly (disassembly) machines take different components (part) from one or more input buffer(s) and produce a single part (parts of different types) which is (are) placed in a downstream buffer (different output buffers).
- Split/merge systems, where different part types are managed. A split machine receives all parts from a single upstream buffer and then places the processed parts in different downstream buffers according to the part type and the adopted policy. A merging machine receives the parts of different type from more than one upstream buffer, but places all the processed parts in a single downstream buffer. The split/merge systems differ from the assembly/disassembly ones because parts cannot be modeled as components of the same product. For further details please refer to [2].
- Closed loop systems, where the last machine of the system is linked with the first machine and the number of parts in the system remains constant.
- General production lines with dedicated (flexible) machines.

The simulation of these classes of manufacturing systems via a distributed approach is strongly related to the representation of the part and information flow between the subsystems, thus rising the need to formalize this kind of transfer. In literature [11] the part transfer was formalized by the definition of the *entity passing* problem where the term *entity* refers to elements that are dynamically created and moved during a simulation [9]. The main result in the formalization of entity passing problem was presented by CSPI-PDG with the definition of Type A IRM, namely Entity Transfer.

Fig. 1 outlines the basic idea behind both Type A.1 and Type A.2 IRMs. The manufacturing system is decomposed into subsystems consisting of workstations and buffers, and each subsystem is associated with a different simulation model. M_i represents the model of the *i-th* production subsystem and En_{ij}/Ex_{ij} represent the *j-th* entry/exit point in M_i. Q_{ik} is associated with the *k-th* buffer in M_i, W_{ih} stands for the *h-th* workstation in M_i, whereas the arrows represent the flow of entities. In Fig. 1, an entity can be transferred from workstation W_{1a} to buffer Q_{2a}.

Type A.1 IRM (named "General Entity Transfer") is defined as the transfer of entities from one model to another, i.e. an entity leaves from a given place in a sending model (e.g. M_1) at time T_1 and arrives at a given place in a receiving model (e.g. M_2) at time T_2 ($T_1 <= T_2$). The departure and arrival places can be buffers, workstations, etc. (e.g. W_{1a} is a departure place and Q_{2a} is an arrival place). According to Type A.1 IRM the entity transfer is always feasible, thus no authorization is required before passing an entity from a model to another. The entity transfer represented in Fig. 1 can be of Type A.1 IRM only if the capacity of Q_{2a} is unbounded.

Type A.2 IRM (named "Bounded Receiving Element") is defined as the relation between a generic element in the sending model and a bounded element in receiving model, so that the feasibility of the entity transfer depends on the state of the receiving element. For instance, an entity transfer is blocked when the queue of the receiving element is full, even if the entity is ready to leave at time T_1 and would attempt to arrive at the bounded element at time T_2. Therefore, the information about the target element state is needed by the sending model, since it could stop the entity transfer. The entity transfer represented in Fig. 1 is of Type A.2 IRM if the capacity of Q_{2a} is bounded.

Type A.3 IRM (named "Multiple Input Prioritization") represents the case where an element of the receiving model can receive entities from more than one sending model. A problem arises if entities coming from different places arrive at the same time (i.e. there are simultaneous events) and the receiving element is not able to receive all of them [12, 17].

III. THE PROBLEM OF ENTITY TRANSFER

Past research contributions presented solutions to face both Type A.1 and Type A.2 IRMs. In particular, a reference architecture (Fig. 2) for the distributed simulation [12] and a protocol to manage the communication between the architecture components [9, 11, 12] were proposed.

Section III.*A* presents the reference architecture adopted in this work and highlights the open issues, whereas Section III.*B* introduces the communication protocols and highlights their major drawbacks.

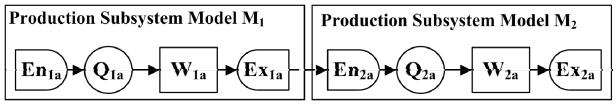


Figure 1. Entity Transfer

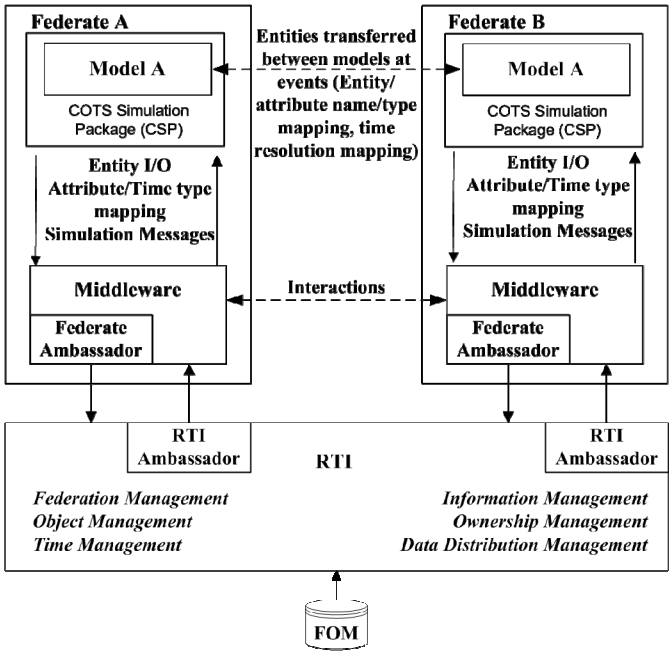


Figure 2. Reference Architecture

A. Reference Architecture

The general architecture shown in Fig. 2 is taken as a reference throughout this work and is mainly based on architecture proposed by Taylor et al. [12]. A detailed description of the architecture components can be found in [11] and [12].

Each federate consists of a COTS simulation package (CSP), a model that is executed by the CSP, and the middleware that is a sort of adaptor interfacing the CSP with the Run Time Infrastructure (RTI). The relationship between CSP, the middleware and the RTI consists of two communication flows: (1) middleware-RTI, (2) CSP-middleware. The middleware *translates* the simulation information in a common format so that the RTI can share it with the federation. In addition, the middleware *receives* and *sends* information from/to the CSP.

Two main issues arise when the simulation information is *translated* for the RTI:

- A common *time definition* and *resolution* is necessary. For example, the *time* should be defined as being the time when an entity exits a source model and then instantaneously arrives at the destination model (i.e. the definition of time implies zero transit time) [11].
- The *representation* of an *entity* depends on how the simulation model is designed and implemented in a CSP. Indeed, the names that the modelers use to represent the same entity might be different. A similar problem can arise for the definition of simple datatypes. For example, some CSPs use 32-bit real numbers while others use 64-bit [11].

Since the aforementioned issues are related to the adopted CSP and the decisions taken by the modelers, two simplifying hypotheses have been made in this paper: all the models have the same time definition and resolution (i.e. there is a relationship between how time is represented in one CSP and another) and a mapping relationship exists between the entity representations in the various models.

This paper addresses the transfer mechanism that is used to move an entity from one model to another thanks to the communications between middleware and RTI, and between CSP and middleware. CSPI-PDG already proposed the Entity Transfer Specification (ETS) Protocol [11, 12] to manage communication at middleware-RTI level (see Section III.*B*). Herein a communication protocol based on Simulation Messages (see Fig. 2) is proposed to manage the communication between a CSP and its middleware (or adapter) when simulating a network of Discrete Event Manufacturing Systems that is characterized by the transfer of parts in the presence of buffers with finite capacity (see Section IV.*B*). The presence of Simulation Messages is the main difference between the reference architecture in Fig. 2 and the architecture proposed in [12].

B. ETS Protocol

The ETS protocol proposed by CSPI-PDG defines the communication between the sending model and the receiving model (*ModelA* and *ModelB* in Fig. 2, respectively) at RTI level by means of a special hierarchy of interaction classes. An interaction class is defined as a template for a set of characteristics (parameters) that are in common within a group of interactions [4]. The middleware of the sending model instantiates a specific interaction class and sends it to the RTI whenever an entity has to be transferred.

After developing the interaction class hierarchy, following the HLA standard, the extensions to Object Model Template (OMT), Simulation Object Model (SOM) and Federation Object Model (FOM) were proposed by CSPI-PDG to include the novel interactions and their parameters. In particular extensions were proposed to the Interaction Class Table to include the novel interaction classes and define them as publish and/or subscribe. The Parameter Table was modified to include the proposed parameters for the interactions and the Datatype table was also modified. For further details please refer to [12].

Straßburger [9] highlighted some relevant drawbacks in the ETS standard proposal:

- It is not possible to differentiate multiple connections between any two models.
- ETS suggested interaction hierarchy does not work: a federate subscribing to the superclass will never receive the values transmitted in the interaction parameters.
- The specification of user defined attributes is placed into a complex datatype, this introduces new room for interoperability challenges as all participating federates have to be able to interpret all of the attributes.

- There are some possibilities for misinterpretation in the definition of "Entity" and "EntityType" introducing changes in FOMs whenever a new entity type is talked about.

Furthermore, the ETS was not designed to manage the Type A.2. IRM and the interaction class hierarchy refers to the entity transfer without taking into account any information on the state of the receiving buffer (e.g. Q_{2a} and Q_{2b} in Fig. 3).

The industrial cases defined in Section II can be modeled only if the first drawback of the list is properly addressed and if the ETS draft standard is modified to manage Type A.2. IRM as well. Indeed, the simulation of a manufacturing system in a distributed way may ask for the representation of multiple connections between the models, thus requiring multiple entry points in a receiving model (e.g. *Model 2* in Fig. 3) and/or multiple exit points in a sending model (e.g. *Model 1* in Fig. 4).

IV. A SOLUTION PROPOSAL FOR THE TYPE A.2 IRM

This section presents the solutions proposed to cope with the problems related to the Type A.2 IRM, as highlighted in Section III. In particular, Section IV.*A* proposes a modification to ETS, whereas Section IV.*B* describes a new protocol aimed at managing the communication between a CSP and its middleware.

A. Proposal to modify Entity Transfer Specification

The ETS standard proposal [11, 12] is modified by defining a new class hierarchy. In particular, different subclasses of the *transferEntity* superclass are defined to face the drawbacks presented in Section III.*B* (Fig. 5).

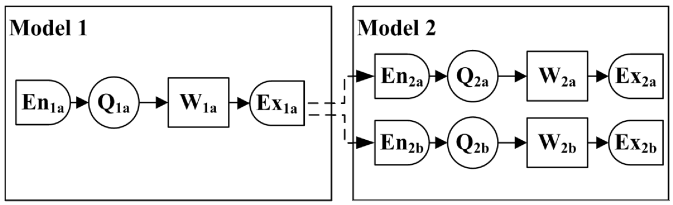


Figure 3. Multiple Part Type Production System (e.g. disassembly or split system) - *Case I*

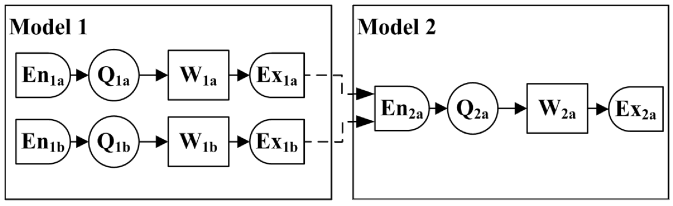


Figure 4. Multiple Part Type Production System (e.g. assembly or merge systems) - *Case II*

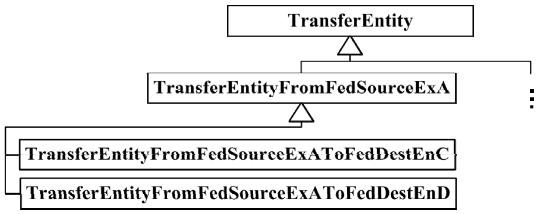


Figure 5. Interaction Class Hierarchy

The resulting class hierarchy consists of the following classes:

- *transferEntity,* as already defined in the ETS protocol. This superclass allows the federate subscribing to all the instances of entity transfer. The instantiation of this class is related to visualization and monitoring tasks.
- *transferEntityFromFedSourceEx* is a novel subclass defined for every exit point, where *FedSourceEx* stands for the name or abbreviation of a specific exit point in the sending model. This class is useful to group the instances of the *transferEntity* that are related to the source federate, so that the *FedSourceEx* can subscribe to all these instances without explicitly naming them.
- *transferEntityFromFedSourceExToFedDestEn* is a novel subclass defined for every pair of exit point (*Ex*) of the source federate (*FedSource*) and entry point (*En*) of the receiving federate (*FedDest*). This class is instantiated both when a sending model needs to transfer a part to a specific entry point in the receiving model, and when a receiving model needs to share information about a buffer state or about the receipt of a part from a specific exit point in a sending model. The models both publish and subscribe to this subclass that was designed to create a *private communication channel* between the sending and the receiving models. Therefore, if an entry point in the receiving model is connected with multiple federates/exit points, then the receiving federate has to inform about the state of the entry point by means of multiple interactions, each dedicated to a specific federate/exit point. This communication strategy is not the most efficient in a generic case, but it offers the possibility to deliver customized information and adopt different priorities for the various federates/exit points.

The ETS Interaction class table was modified to represent the *transferEntityFromFedSourceEx* and *transferEntityFromFedSourceExToFedDestEn* subclasses, whereas the Parameter Table was modified to include the parameters of the novel interaction class *transferEntityFromFedSourceExToFedDestEn*.

The parameters in the Parameter Table are defined as follows and the similarities with the parameters included in the ETS Parameter Table [12] are highlighted where present.

- *Entity*. It is a parameter of complex datatype containing the *EntityName* that is used to

communicate the type of the entity, and the *EntityNumber* that is used to communicate the number of entities to be transferred. The *EntityName* and *EntityNumber* play the role of the *EntityName* and *EntitySize* defined in ETS [11, 12], respectively.

- *ReceivedEntity*. It refers to the entity received by the receiving federate and has the same type of the parameter *Entity*.
- *Buffer_Availability*. It was defined to enable the communication about the buffer state.
- *SourcePriority*. This parameter was defined to communicate the priority assigned to the entity source, so that the infrastructure can be further extended to manage Type A.3 IRM.
- *EntityTransferTime*. It defines the simulation time when the entity transfer starts. This time is equal to the entity arrival time, since it is assumed that the transferred entity instantaneously arrives at destination (see the definition of time in Section III.A). The transit time is not explicitly modeled, but it can be taken into account if *EntityTransferTime* represents the entity arrival time, thus considering the transit time as well.

B. Simulation Messages

Simulation Messages are designed to support the communication between a CSP and its middleware (see Section *III.A*) that is not on the HLA side. The choice of a communication protocol depends on the role played by the federate. In the case of sending federate the protocol manages the communication concerning the need of sending an entity to another model (outgoing communication) and/or the possibility of sending the entity outside the model (incoming communication). In the case of receiving federate the protocol manages the communication concerning the buffer state and/or the acceptance of an entity (outgoing communication) and/or the receipt of an entity from other models (incoming communication). Simulation Messages are implemented as a class which is characterized by the following attributes:

- *time* that refers to the simulation time when the message is sent to the middleware from the CSP. This attribute is used by the middleware to determine the *TimeStamp* of the interaction that will be sent to the RTI.
- *BoundedBuffer* that contains the information about the state of the bounded buffer in the receiving model.
- *TransferEntityEvent* that represents the entity transfer event scheduled in the sending model event list and contains the information about the entity to be transferred and the scheduled time for the event.
- *ExternalArrivalEvent* that represents the external arrival event that is scheduled in the receiving model. It contains the information about the entity to be received and the scheduled time for the event.
- *ReceivedEntity* that represents the information about the entity that was eventually accepted by the receiving model.

The CSP of the sending federate sends a message to its middleware whenever a *TransferEntityEvent* is scheduled (i.e. the departure event of an entity from the last workstation of the sending model is added to the simulation event list). Then, the middleware uses the attributes *time* and *TransferEntityEvent* to inform the RTI about the need of passing an entity, while the simulation keeps on running. When the simulation time arrives at the *TransferEntityEvent* time (that corresponds to the *EntityTransferTime* presented in Section IV.A) and after all the local events scheduled for that time instant have been simulated, the request for the advance to *EntityTransferTime* is sent by the middleware to the RTI. After the time has advanced, the middleware informs the CSP of the sending model about the state of the receiving buffer in the receiving model. If the receiving buffer is not full, then the workstation can simulate the *TransferEntityEvent*, otherwise it becomes blocked. From the blocking instant until when the middleware informs that the receiving buffer is not full, the model keeps on sending requests for time advance at the lookahead value. The CSP of the receiving federate sends a message to its middleware whenever a change in the buffer state occurs and this message contains the updated value of the attribute *boundedBuffer* representing the state of the buffer. Then, the middleware communicates to the RTI the state of the buffer via interactions. If the change in the buffer state is due to the arrival of an entity from another model, then the update of the information does not imply *zero lookahead* and the communication is characterized by defining the entity that has been accepted (i.e. the *ReceivedEntity* attribute). If the buffer state change is not related to an external arrival, then the update of the buffer information implies a *zero lookahead* [12] whenever it is not possible to determine an advisable *a-priori* lookahead for the federation. After being informed by the middleware that another federate needs to transfer an entity, the receiving model actually simulates the arrival of the entity only if the buffer is not full, otherwise the arrival is not simulated and the workstation in the sending model becomes blocked.

The application of the Simulation Messages can be better appreciated by describing an example that is characterized as follows: (1) the reference production system is represented as in Fig. 1, (2) the buffer Q_{2a} at time t accommodates a number of parts that is greater than zero and lower than the buffer capacity and an entity enters workstation W_{1a}, (3) a departure event from workstation W_{1a} is scheduled for time $t' = t + p$, where p represents the processing time of the leaving entity at station W_{1a}, (4) during the time interval (t, t'), no event happening in the federate M_2 (local event) influences the state of the buffer Q_{2a}.

Since W_{1a} is the last machine in model M_1, the departure event is also a *TransferEntityEvent*. Therefore, the CSP sends a message to its middleware containing *time* (t) and *TransferEntityEvent* attributes. After receiving the message, the middleware of the sending model informs the RTI via interaction. When the RTI time advances to time t, the middleware of the receiving model receives the information

about the need of the sending model to transfer an entity at time t'. Then, the middleware sends to the receiving model a simulation message containing the *ExternalArrivalEvent*. The receiving model simulates the external arrival as soon as the simulation time advances to t' and all local events for that time have been simulated (since the buffer Q_{2a} is not full according to the example settings). A message is sent to the middleware of the receiving model containing the updated values of the Q_{2a} state (attribute *BoundedBuffer*) together with the information concerning the recently accommodated part (attribute *ReceivedEntity*). Afterwards, the middleware sends two interactions to the RTI: one is with a *TimeStamp* equal to t' and contains the updated state of the buffer Q_{2a} and the receipt of the entity, the other contains the request of time advance to time t'. When the RTI advances to time t', the middleware of the sending model receives the information regarding the state of Q_{2a} and the received entity by means of the RTI. Since the entity has been delivered to the receiving model, the station W_{1a} is not blocked by the middleware.

V. HLA-BASED INFRASTRUCTURE IMPLEMENTATION

The HLA-based architecture shown in Fig. 2 was implemented as follows:

- MAK-RTI 3.3.2 [6] was used as the RTI component implementation.
- The middleware was developed in C++ language following the specifications defined in Section IV and was named *SimulationMiddleware*.
- The simulation models were developed using a CSP emulator, thus following the approach suggested in [17].

The *FederateAmbassador* and *RTIAmbassador* were provided by MAK-RTI as C++ classes and were linked to the *SimulationMiddleware*. Further extensions were needed to implement the proposed modification to ETS (see Section IV.*A*) and the Simulation Messages (see Section IV.*B*). The former required a modification to *FederateAmbassador* class, whereas the latter led to the development of a new C++ class. The *SimulationMiddleware* was implemented to manage the information contained in Simulation Messages.

VI. MANUFACTURING PRODUCTION SYSTEM: A TEST CASE

The experiments presented in this section were designed to test the proposed entity passing solution (Section IV.*A*) .The test case refers to a factory that produces two part types (*Part_A* and *Part_B*) and consists of two separated manufacturing systems (*Plant1* and *Plant2*). *Plant1* executes a set of manufacturing operations on both part types, whereas *Plant2* is divided into two production lines and each line is dedicated to process a single part type. If the performance of *Plant1* and *Plant2* is simulated via two separated simulators, then the whole production system can be simulated in a distributed way by representing *Plant1* and *Plant2* as federates within a federation (see Fig. 6).

This test case is characterized by Type A.2. IRM, because the workstation W_{12} has to transfer parts of *type A* (*type B*) to the bounded buffer Q_{21} (Q_{23}) in the *Part_A Line* (*Part_B Line*) of *Plant2*. For validation purposes the results from the distributed simulation were compared to a monolithic reference implementation. Two simulators representing *Plant1* and *Plant2* were developed using the CSP emulator mentioned in Section V. These simulators were integrated by means of the HLA-based infrastructure (*DS approach*). Furthermore, a monolithic simulator was built using the CSP emulator to represent the overall factory (*SA approach*). Finally a monolithic simulator was built using Rockwell Arena® 12 to validate the CSP emulator (results of this validation are not reported since they are out of scope). It is assumed that all simulation models do not contain any stochastic element. The experiments were designed as follows:

- Parts of type *A* arrive at *Plant1* every two time units, whereas parts of type *B* every time unit.
- The capacity of Q_{11} was set to a value large enough to guarantee that it never becomes completely full.
- Three conditions of maximum capacity (i.e. 1, 5, 50) were considered for the other buffers.
- Three conditions of processing times were considered for the workstations. Equal to 1 for all the workstations in the first condition, then equal to 7 for workstation W_{11} and to 1 for all the other workstations in the second condition, and finally equal to 7 for workstations W_{21} and W_{23} and to 1 for all the other workstations in the third condition.

Nine experimental conditions were obtained by combining the buffer capacity and processing time conditions. For each experimental condition the performance of the manufacturing system was simulated according to both the *SA* and the *DS* approach. A simulation length of 10000 time units was set for all the experiments. A conservative synchronization approach was adopted, given the deterministic nature of the simulation, it was possible to evaluate the smallest interval elapsing between two consecutive events and the lookahead was set to that value (i.e. 1 time unit) for all the experiments. The experiments were run on a single machine Intel Core2 Duo Processor T7250, 2.00 GHz and 2046 MB-RAM.

Both simulation approaches (*DS* and *SA*) returned identical simulation output values for each experiment, thus validating the interoperability offered by the proposed solution for Type A.2 IRM. However, the *DS* approach required a significantly higher (four times on average) computational time compared to the *DS* approach because of the overhead related to the services offered by RTI. The aforementioned result led to examine the number of interactions sent during the simulation experiments, thus delving into the overhead related to the RTI services.

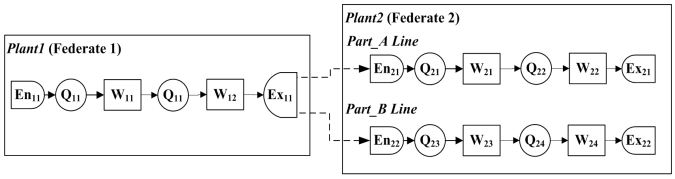


Figure 6. Factory Test Case

Table I reports the number of interactions sent during the distributed simulation of the experiments characterized by the first condition of buffer capacity and the third condition of processing times. This experimental condition is the most critical in terms of the number of interactions needed to synchronize the two federates because the two slowest workstations are placed in *Plant 2* and therefore buffers Q_{21} and Q_{23} are frequently full while workstation W_{12} is frequently blocked. As a consequence, many interactions are needed to communicate when the parts can be actually transferred according to the state of buffers Q_{21} and Q_{23}.

The receiving federate sends more entity passing interactions than the sending federate because the receiving federate sends an interaction every time an entity is received and/or its state is changed, whereas the sending federate sends an interaction only when a departure event is scheduled. Each part arriving at the buffers, except the first that is directly assigned to the machine, causes two interactions: one to communicate the receipt of the entity and the updated buffer state, and another one to communicate the updated buffer state when the entity leaves the buffer to enter the next workstation. A large number of time interactions is necessary because a request for time advance is generated at every time unit since the workstation W_{12} is almost always blocked and the time advances at the lookahead. The number of time advance interactions does not decrease if the slowest machine is placed in *Plant1*, since the receiving federate is still not aware of when the entity will arrive. This behavior highlights how the solution proposed in Section *IV* needs to be further improved.

VII. INDUSTRIAL CASE

This section aims at verifying if the proposed integration infrastructure presented in Section III can help to better evaluate the performance of complex manufacturing systems as described in Section II. The goal does not consist in comparing the HLA-based distributed simulation with a monolithic simulation in terms of accuracy and computational efficiency as already done in Section VI. Herein, the attention is focused on the industrial field represented by sheet metal production, thanks to the collaboration offered by the company Tenova Pomini. In this industrial field, the production systems are characterized by the presence of at least two subsystems interacting with each other (Fig. 7). The *Roll Milling system* produces sheet metal using milling rolls that are subject to wearing out process; once the rolls wear out (i.e. at the end of the *roll life*) there is the need to change them to avoid defects in the laminates. The *Roll Shop* performs the grinding process to recondition the worn out rolls. Tenova Pomini is a designer and provider of *RollShop* systems.

TABLE I. ANALYSIS OF INTERACTIONS

Type of Interaction	**Total Number of interactions**	
	Sending Federate	Receiving Federate
Time Advance	9999	10000
Entity Passing (part type *A*)	715	1429
Entity Passing (part type *B*)	1429	2857

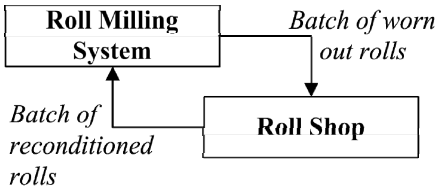


Figure 7. Industrial Case representation

If the attention is focused on the rolls, then the resulting production system can be considered as a closed loop: the Roll Milling system sends batches of worn out rolls to the Roll Shop following a given policy and receives reconditioned rolls back.

The two subsystems forming a closed loop are strongly related and their reciprocal influences should be considered to properly evaluate the performance of the whole factory. However, the lack of shared information between the owner of the Roll Milling system and the Roll Shop designer makes the realization of a monolithic simulation model hard to obtain or even infeasible. In particular, the Roll Milling system works according to specific *roll changing policies* that are not shared with the Roll Shop designer even if these policies play a key role in the dynamics of the whole factory. Indeed, when a roll is worn out, the remaining life of the other rolls is checked and if the remaining life of a roll is under a predefined threshold, then it is sent to the grinding system together with the completely worn out rolls. Therefore, these policies determine a relation between different roll types, since a roll can be sent to the grinding system depending on the behavior of other roll types.

When Tenova Pomini designs a Roll Shop, the owner of the Roll Milling system provides aggregated information about the yearly average demand of worn out rolls to be reconditioned. Then the Roll Shop designer develops a simulator for the roll grinding process with high level of detail.

The hypothesis is made that the Roll Shop designer has developed and validated a simulator using the CSP emulator (see Section *V*). Similarly, the Roll Milling system owner has developed and validated a simulator using the CSP emulator, modeling the milling process and the roll changing policy with high level of detail; however, the model of the Roll Milling system simulator is not shared with the Roll Shop designer.

The *Service Level* (*SL*) is the typical key performance indicator (KPI) for evaluating the Roll Milling system and is defined as the ratio of the time during which the milling system is producing laminates over the total time when the milling system is available (i.e. there is no failure).

The *Service Level* would be reduced if the Roll Milling system had to wait for reconditioned rolls coming from the Roll Shop.

The Roll Shop designer has to evaluate the system performance in terms of *SL* while taking into account the influence of the Roll Milling system related to:

- The arrival rate of worn out rolls from the Roll Milling system that is estimated from the yearly aggregate demand of reconditioned rolls.
- The acceptance of the reconditioned rolls sent by the Roll Shop (closed loop model).

The influence of the Roll Milling system can be modeled by adding a virtual workstation inside the detailed simulation model of the Roll Shop. This virtual workstation roughly models the Roll Milling system by generating the arrival of worn out rolls and accepting the reconditioned ones. Therefore, the *Service Level* can be calculated with reference to the performance of the virtual workstation. The realization of a simulation model like this will be referred to as *Approach A*. The main drawbacks of *Approach A* consist in:

- The real behavior of the Roll Milling system cannot be precisely modeled since it is reduced to a black box sending and receiving rolls (e.g. the roll changing policies are not modeled).
- The performance (e.g. mean starvation time for every station, mean level of roll buffers, etc) of the Roll Milling system cannot be evaluated.

These drawbacks lead to a potentially inaccurate evaluation of the factory performance. The Roll Shop designer could increase the level of detail of the virtual workstation to improve the completeness of its simulation model and the accuracy of the estimated *Service Level*. However, the lack of shared information hinders the feasibility of this enhancement. It must be stressed that simulator of *Approach A* cannot be considered as a proper monolithic simulator of the whole factory, since the Roll Milling system is only poorly modeled.

An alternative approach (*Approach B*) to evaluate the performance of the whole factory can be developed by adopting the proposed HLA–based Infrastructure to integrate the simulators of the Roll Milling system and of the Roll Shop. In this case, the virtual workstation is removed from the simulation model of the Roll Shop, since the behavior of the Roll Milling system is already modeled by its simulator. *Approach B* enables to evaluate the impact of the number of rolls on the system performance, so that the Roll Milling System owner can optimize the investment cost associated with the expensive rolls, whereas the Roll Shop designer can design a more effective and efficient system thus better meeting the needs of the customers.

The two approaches have been compared by designing a set of experiments that are characterized as follows:

- Three experimental conditions are designed with reference to the total number of rolls circulating in the whole system. These three conditions are defined as *Low*, *Medium*, *High* level.
- The simulation run length was set to six months.
- The roll changing policy adopted for the *Approach B* simulator has been kept fixed throughout the experimentation.

The results of the experiments are reported in Table II.

TABLE II. SERVICE LEVEL RESULTS

Experimental Conditions	*Approach A*	*Approach B*	*Percentage difference*
High Level	0.995	0.872	12.3
Medium Level	0.946	0.682	27.7
Low Level	0.308	0.273	3.5

Approach A and *Approach B* are compared in terms of the estimated *Service Level*. The results show that the difference between the two approaches is larger for the *High* and *Medium* level conditions. If the level of rolls is *Low* then the roll changing policy does not affect the overall performance of the production system because the Roll Milling system is frequently waiting for reconditioned rolls and therefore the estimations are similar. In case of *Medium* and *High* level conditions the workload of the rolls in the Roll Shop can be strongly influenced by the roll changing policy, thus generating a higher difference in the estimation between the two approaches.

Based on the analysis carried out so far, it was decided to design further experiments to analyze the behavior of the system with different starting workload conditions, i.e. the number of rolls that are present in the Roll Milling system when the simulation starts.

These experiments can be useful to analyze the ramp-up period and select the roll changing policy that avoids the arising of critical workload conditions.

These additional experiments can be carried out only adopting *Approach B*, since the starting workload conditions cannot be modeled with *Approach A*. Indeed, the virtual workstation generates rolls for the Roll Shop independently from the starting workload conditions. Therefore, the virtual workstation would generate roll arrivals even if all the rolls are already located in the Roll Shop, thus incorrectly increasing the number of rolls in the whole factory. This represents an additional criticality of the *Approach A* that can be solved only using *Approach B*.

The second set of experiments was designed as follows:

- Two *types of roll* circulate in the factory (*RollType1* and *RollType2*). The roll of type *RollType2* has a longer roll life than *RollType1*.
- For each type of roll three levels of the *Starting Workload* (i.e. number of rolls) in the Roll Milling system are considered.
- Three simulation run lengths are considered, i.e. 1 week, 2 weeks and 4 weeks.
- The roll changing policy is fixed for all experiments.
- The total number of rolls is equal to the *High* level of the previous experimentation and is fixed for all the experiments.

Fig. 8 shows the main effects plot for the *Service Level* evaluated by simulating the 27 resulting experimental conditions with *Approach B*.

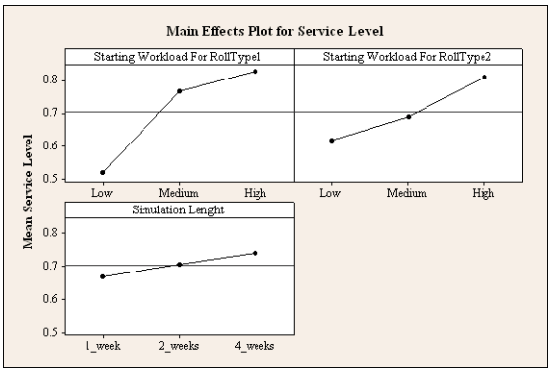


Figure 8. Main Effect Plot for *Service Level* with *Approach B*

The plot suggests a significant influence of the factor *Starting Workload for RollType1*. This roll type assumes a key role because of its short roll life. If the *Starting Workload for RollType1* is *Low*, then the Roll Shop can hardly cope with the frequent roll requests of *RollType1* from the Roll Milling system during the transient period and low values of *SL* are observed. This transient phenomenon occurs in all conditions of the simulation lengths, but it mitigates when the simulation length increases. Indeed the *SL* tends to a stationary value that is independent from the starting conditions. Nonetheless this analysis can be useful for the Roll Milling system owner that can individuate critical conditions, thus designing roll changing policies that avoid the occurrence of these situations during the ramp-up phase.

VIII. CONCLUSIONS AND FUTURE WORKS

The need of simulating complex manufacturing systems led to investigate the integration of CSPs based on HLA and to propose a solution to the CSP interoperability problem by addressing the Type A.2 IRM and modifying the ETS protocol. In particular, Simulation Messages were designed to manage the communication between a CSP and its adaptor in case of Type A.2 IRM. Nonetheless, the implementation can be extended to the case of simultaneous events (Type A.3. IRM). The experiments showed the feasibility of the use of the integrated simulators infrastructure, but further developments are needed to optimize the time advance management. The effect of different lookahead values should be evaluated together with the use of optimistic synchronization approaches. The analysis of the number of interactions needed for the entity transfer (see Section VI) highlights that further research on the implementation of Type A.2 IRM is necessary as well. For instance, it would be interesting to investigate the design of protocols that do not force to send interaction at every time unit to communicate the state of the federates, but enable the interaction depending on the system state. Finally Section VII showed how the HLA-based distributed simulation can be exploited in the industrial domain.

ACKNOWLEDGMENT

The research reported in this paper has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement No: NMP2 2010-228595, Virtual Factory Framework (VFF). The authors would like to thank Prof. S.J.E. Taylor for his on-going scientific support and TENOVA POMINI for the definition of the industrial case.

REFERENCES

[1] Boer, C.A. 2005. Distributed simulation in industry. *PhD thesis. Erasmus Research Institute of Management (ERIM), Erasmus University Rotterdam, The Netherlands*.

[2] Colledani, M., T. Tolio. 2005. A Decomposition Method to Support the Configuration/Reconfiguration of Production Systems. *CIRP Annals - Manufacturing Technology*, vol. 54, Issue 1, pp. 441-444.

[3] Hibino, H., Y. Fukuda, Y. Yura, K. Mitsuyuki, K. Kaneda. 2002. Manufacturing Adapter Of Distributed Simulation Systems Using HLA. In *Proceedings of the 2002 winter simulation conference E. Yücesan, C.H. Chen, J. L. Snowdon, and J. M. Charnes, eds*, vol. 2, pp. 1099 - 1107.

[4] IEEE 1516. 2000. Standard for Modeling and Simulation (M&S) High Level Architecture (HLA).

[5] Law, A.M., 2007. Simulation modeling and Analysis, 3rd ed. New York: McGraw-Hill.

[6] MAK RTI, www.mak.com.

[7] McLean, C., F. Riddick. 2000. Integration of manufacturing simulations using HLA. In *Proceedings of the 2000 Advanced Simulation Technologies Conference (ASTC 2000).*

[8] SISO CSPI-PDG www.sisostds.org.

[9] Straßburger, S. 2006a. The road to COTS-interoperability: from generic HLA-interfaces towards plug-And-play capabilities. In *Proceedings of the 2006 Winter Simulation Conference*, *L.F. Perrone, F.P. Wieland, J. Liu, B.G. Lawson, D.M. Nicol, and R.M. Fujimoto*, *eds*., pp.1111 - 1118.

[10] Straßburger, S. 2006b. Overview about the high level architecture for modeling and simulation and recent developments. In *Simulation News Europe* 16(2), pp. 5-14.

[11] Taylor, S.J.E., S. Turner, and M. Low. 2004. A Proposal for an Entity Transfer Specification Standard for COTS Simulation Package Interoperation. In *Proceedings of the 2004 European Simulation Interoperability Workshop*. June 28 - July 1, 2004. Edinburgh, Scotland.

[12] Taylor, S.J.E., Wang, X., Turner, S.J., Low, M.Y.H. 2006. Integrating Heterogeneous Distributed COTS Discrete-Event Simulation Packages: An Emerging Standards-Based Approach. IEEE Transactions on Systems, Man & Cybernetics: Part A, 36, 1, pp. 109-122.

[13] Taylor, S.J.E., N. Mustafee, S. Turner, K. Pan, S. Straßburger. 2009. Commercial-off-the-shelf Simulation Package Interoperability: issues and futures. In *Proceedings of the 2009 Winter Simulation Conference M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin and R. G. Ingalls, eds*, pp. 203 - 215.

[14] Terkaj, W., T. Tolio and A. Valente. 2009. Designing Manufacturing Flexibility in Dynamic Production Contexts. In *Design of Flexible Production Systems Methodologies and Tools*, Springer. DOI 10.1007/978-3-540-85414-2. pp. 1-18.

[15] Vàncza, J., P. Egri, L. Monostory. 2008. A coordination mechanism for rolling horizon planning in supply networks. *CIRP Annals - Manufacturing Technology,* Volume *57*, 455-458.

[16] Sacco, M., Pedrazzoli, P., Terkaj, W. 2010. VFF: Virtual Factory Framework. In *Proceedings of 16th International Conference on Concurrent Enterprising*, Lugano, Switzerland.

[17] Wang, X., S.J. Turner, S.J.E. Taylor. 2006. COTS Simulation Package (CSP) Interoperability – A Solution to Synchronous Entity Passing. In *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation (PADS'06),* pp. 201 - 210.

[18] Wiendahl, H.P., S. Lutz. 2002. Production in Networks. *CIRP Annals - Manufacturing Technology,* vol. *51*, pp. 573-586.

Support System for Distributed HLA Simulations in Industrial Applications

Michael Raab, Steffen Masik
VIT
Fraunhofer IFF Magdeburg
Magdeburg, Germany
michael.raab@iff.fraunhofer.de
steffen.masik@iff.fraunhofer.de

Thomas Schulze
FIN/ITI/UMS
Otto-von-Guericke University
Magdeburg, Germany
thomas.schulze@ovgu.de

Abstract— **Coupling and synchronized execution of legacy simulation models is a complex task. Simulation experts need knowledge in the domains of distributed simulation and HLA to conduct such complex simulation projects. This is one reason for insufficient acceptance of distributed simulation in industrial areas. To increase the number of distributed simulation projects in industrial domains, a dedicated access for industrial engineers to this technology is needed. Therefore it is necessary to develop and apply support systems for distributed simulations with user-interfaces that are dedicated to factory planners and operators. The objective of this paper is to discuss the provision of distributed simulation support systems for in industrial applications in more detail.**

Distributed Simulation; manufacturing application; support system

I. MOTIVATION

Simulation is an established tool for the planning and operation of complex production and logistics systems. Differently sized and detailed simulation models are created with various simulation software systems depending on the intended application. Complex production systems are often divided into different sections or components and typically different simulation models are developed to represent each section or component. While the real sections are interdependent, the corresponding simulation models are often independent of each other. For example in automotive industry, it is not unusual to model and simulate the main factory sections car body, paint shop and final assembly separately, each having their own system boundaries and no interconnection. In reality, these areas of a factory are of course interconnected. The car body section needs to respond to changing conditions in the paint shop, e.g. the occurrence of a break, a failure or some kind of service time. These connections between the sections of a real factory are often not or even inadequately modeled in the corresponding simulation models. Conclusions about a complex production system's performance inferred from different simulation models are biased by the absence of interconnectedness of the simulated sections.

One solution to improve this situation is to develop a new monolithic factory model. This approach requires considerable modeling time and the created model generally has a lower level of detail than the related pre-existing section models. Another solution is to connect existing legacy simulation models to a new, more complex model network. This entails coupling either homogenous or heterogeneous models. Some object-oriented simulation systems such as Tecnomatix Plant Simulation support the former. However, many real industrial applications need coupled heterogeneous models.

Distributed simulation technology in general permits the coupling of heterogeneous simulation models. Originally investigated in the military domain, distributed simulation was initially developed to provide a more effective way to train personnel in distributed virtual environments that simulate combat situations. The objectives of the application of distributed simulation in the industrial domain can be divided into the categories of speedup (or performance in general) and interoperability. The objective of interoperability is to provide and facilitate collaboration between and reusability of heterogeneous simulation models. This is vital in the context of the digital factory.

Syntactical, technical as well as semantically aspects must be considered when carrying out a distributed simulation project. While the HLA constitutes a standard that supports syntactic interoperability, the efficient execution, control, and management of distributed simulation and the attendant models remain a challenge since no applicable standards exist.

Coupling and synchronizing the execution of legacy simulation models is a complex task. Simulation experts well-versed in the domains of distributed simulation and HLA are necessary accomplish such simulation projects. This is one reason for the rare usage of distributed simulation technology in industrial areas. The number of industrial distributed simulation projects will increase once industrial engineers are provided with dedicated access to the requisite technology. This in turn will necessitate the development and application of support systems for distributed simulations, which feature dedicated user-interfaces for factory planners and operators.

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

This paper examines the issues related to support systems for distributed simulations in industrial applications.

The remainder of this paper is structured as follows: Section 2 furnishes a brief overview of research activities related to distributed simulation in industrial applications. Section 3 describes some aspects of the configuration and preparations of legacy simulation models necessary to enable them to participate as federates in a HLA federation. Section 4 discusses some aspects of monitoring during the execution of distributed simulations. Section 5 gives an overview on data acquisition and section 6 presents the architecture used in a developed support system. Section 7 discusses some outcomes of the developed support system. Section 8 concludes the paper and introduces some aspects that deserve more research in the future.

II. Related Research Activities

Much effort has been made to apply distributed simulation to industrial and engineering domains, e.g. [1], [2], [3], [4], [5], [6]. The SISO CSPI product development group (PDG) is additionally working to advance interoperability between different common-of-the-shelf (COTS) simulation packages. This group develops interoperability reference models (IRMs) and proposes reference solutions for their implementation [7], [8], [9]. A recent study of future trends in distributed simulation and distributed virtual environments [8] confirmed that distributed simulation and distributed virtual environments hold high commercial potential. More widespread use of these technologies will be contingent on improved distributed simulation middleware and solutions for semantic interoperability.

Ongoing research activities are primarily focused on the basic information technologies necessary for distributed simulations. Systems that support the preparation and execution of distributed simulations by factory planners and operators of complex production systems have generally received little attention. There are only a small number of published real industrial applications. Arguments again are that benefits are not visible for COTS users as the vendors don't provide integration of distributed simulation aspects into their products [2].

Our general approach is to improve the access for industrial engineers on distributed simulations. A support system is required that guides engineers during the preparation, execution and evaluation of industrial distributed simulations. Existing support tools from distributed environment vendors like Pitch and MAK enable execution of distributed simulations on technical level like monitoring of network and process resources, and management of network security. These tools are missing any user support in terms of semantics. They were built for military and training applications and not for industrial application with simulation of material handling processes. The main requirement for the development of our support system was to support factory planners and operators to apply distributed simulation on technical and semantic level. Thereby the technical aspects will be hidden from the user so that he can focus on the description of the semantic relations between coupled simulation models.

The following is based on experiences gathered from HLA simulation of material handling in the automotive industry.

III. Preparation and Configuration

The execution of distributed HLA simulations can be broken down into the phases of preparation, configuration and monitoring.

Preparation involves the integration of the HLA interface into each simulation model. The interface enables a federate to enter and leave the federation, synchronize the model state and exchange data with other models. The preparation step is executed only once for each participating simulation model. For more details, see [9] and [10]. The implementation of this phase requires dedicated IT skills and generally cannot be performed by operators and planners.

Configuration entails the selection of the legacy simulation models involved and defines their interrelationships. Additional coupling features such as transport systems may have to be modeled and inserted between these legacy models. Such additional models are necessary when, for instance, the material flow between the components of a production system needs to be considered.

Monitoring phase includes the observation of the distributed simulation execution and the collection of data during that execution.

The configuration and monitoring phases are executed sequentially for each simulation experiment, whereas the preparation phase needs to be executed only once. Planners and operators must execute both the configuration and monitoring phases. The necessary steps within these phases are performed (semi-) automatically and should be hidden from users as far as possible. Since the user must specify the experiment design, there is no intention to fully automate the execution phase.

The configuration phase is intended to provide a semantic description of the relationships between the selected legacy models. The following must be specified for each relationship:

- the parts and conditions under which parts are transferred from model M_i to model M_{i+1},
- the transition between the two models and
- the location in model M_{i+1} into which a arriving part will be inserted.

For their experiments, users must specify the parts that are going to be transferred from model M_i to model M_{i+1}. The support system must guarantee that only valid relationships can be specified between models, thus allowing users only to select part types that are needed in model M_{i+1}. Other part types transferrable to models other than M_{i+1} should not be selectable.

Furthermore, the support system must guarantee that parts for model M_{i+1} may only be inserted in dedicated locations and that locations unable to receive parts may not be selected.

The validity of each specified relationship is a general requirement. The support system must read the simulation models or related input data at the start of the configuration

phase. An XML file the complex model's configuration data is written at the end. It allows an abstract description of relationships between material flow simulation models independent of any HLA aspects. The subsequent monitoring phase uses the configuration file as input.

IV. MONITORING

The monitoring phase follows the configuration phase in the execution of a distributed simulation project. Monitoring covers the automatic start and supervision of distributed simulation experiments as well as the collection of additional data that may be used to calculate and present results incorporating the complex distributed simulation model.

Earlier applications of distributed simulations of production systems entailed substantial work on the part of users. Each simulation model needed to be initialized and all models needed be started synchronized. During their run, the user obtained no or only insufficient information on the progress of the simulation and the occurrence of errors. Once run, the user was responsible for drawing conclusions about correlations between models and had to create output objectives for the complex simulation model. Industrial application users need support to perform these operations. The aforementioned steps must be encapsulated and abstracted so that end users are able to focus on the actual task of obtaining new insights into a complex production system.

A. Material Flow Modeling

Monolithic simulation models usually destroy their material flow elements when they reach the end of the modeled process chain. A distributed simulation necessitates the transfer of subsets of these elements to successive models. Our industrial pilot project demands the transfer of material arriving at the end of a model into a bounded buffer of a successor model. The simulation code had to be modified to allow this.

In terms of their interoperability requirements, our distributed simulation models are compliant with a Type A.2 Interoperability Reference Model. This classification is derived from the Standard for Commercial-Off-The-Shelf Simulation Package Interoperability Reference Models [7]. The A.2 reference model type describes the synchronized transfer of entities into a buffer with limited capacity. In our solution implementing this transfer and its conditions, we use a HLA interaction message for the entity transfer and HLA attribute updates to communicate the current buffer content of the respective input buffers. Figure 1. illustrates the principle and some important data structures.

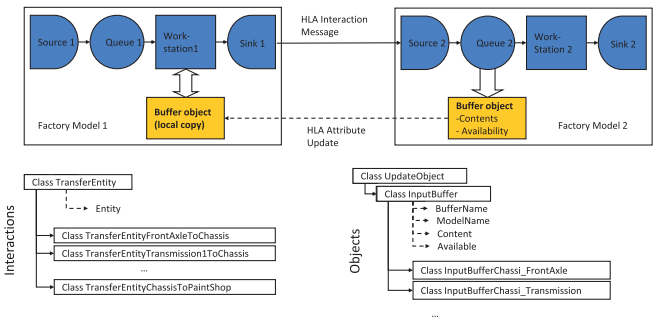


Figure 1. Modeling the material and information flow beyond the boundaries

For allowing the operation in a distributed simulation environment certain information about local objects and their attributes (of interest for the other models) have to be communicated regularly to interested simulation models.

The factory model 1 in Figure 1 needs to know the availability of the input buffer (Queue 2) in factory model 2. Factory model 1 can only transfer a part to factory model 2 when the buffer is available. The buffer object is published to all other models using the HLA object attributes of content and availability. Factory model 2 publishes all changes of these attributes to all other interested simulation models. A local copy of the buffer from factory model 2 is used inside factory model 1. When an attribute update of the original buffer is sent, the local copy is updated automatically. This ensures the consistency between the original and the copy. Factory model 1 makes decisions about part transfer based on the local copy of the buffer in factory model 2. Certain HLA functionalities had to be used to publish, subscribe, send and receive these types of data. These actions were implemented in a manner that is clear to end users.

B. Starting Distributed Simulation

In order to start a distributed simulation, all of the semantic relationships between single simulation models must be configured validly. Local model-specific configuration files for each and every model are derived from the global configuration data of the comprehensive model. These model-specific configuration files specify correlations between the model itself and all directly connected models, predecessors and successors. The created XML configuration file contains information on

- simulation control data, e.g. simulation duration, HLA Lookahead and the path to the federation description file,
- buffers and their attribute values related to the material flow, e.g. name and capacity,
- part types to be sent to successive models, e.g. destination model and destination buffer, and
- part types to be received from predecessor models, e.g. sending model and additional part specific data.

The first part of such example XML-file with simulation control data and buffer values is shown in Figure 2.

```
<Model>
- <General>
  <Name>M1</Name> <Type>TypeA</Type> <SimulationTime>1</SimulationTime>
  <LookAhead>60</LookAhead> <FedFile>C:\Work\...</FedFile>
  <FederationName>FactorySimulation</FederationName>
  </General>
- <Buffer>
  <Name>M1Buffer</Name> <Capacity>10</Capacity>
  </Buffer>
- <PartTypeToSend>
  <PartType>W1</PartType> <Receiver>M4</Receiver> <RecvBuffer>M4O</RecvBuffer>
  </PartTypeToSend>
- <PartTypeToReceive>
  <Sender>M2</Sender> <Buffer>M2In</Buffer> <IncomingPartType>E</IncomingPartType>
  </PartTypeToReceive>
```

Figure 2. Example for simulation control data and buffer attrubutes

Once all local configuration files and the appropriate federation description file (FED file) for the HLA federation

have been generated, the HLA Run Time Interface (RTI) is configured and started. Thereafter, the support system starts each simulation model involved. Figure 3 depicts the launch of two federates. In principle federates can be distributed over more than one computer. In this case a network and some kind of daemon are required.

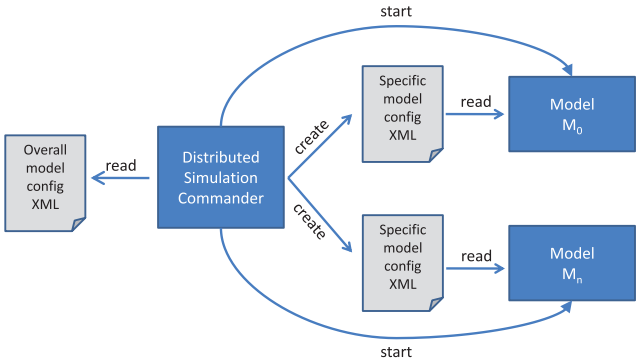


Figure 3. Launch process of two models

Each participating simulation model (federate) joins the federation. All HLA simulation models are generally time-constrained and time-regulating.

All federates must be synchronized at the end of the start phase, i.e. every simulation model must wait until the other federates have joined the federation. We recommend implementing an additional federate called an observer, which has two main functions: synchronizing joining federates and collecting data during a simulation.

During the synchronization phase, the observer blocks all time advances in the federation until all the needed federates have joined the federation. Initially, all federates, including the observer, are both time-regulating and time-constrained. Once all federates have been joined and initialized, the observer terminates its block (switching off time-regulation), and the simulation models start to advance in simulation time. Afterward, the observer federate is time-constrained only and remains passive until the simulation has been run.

C. *Distributed Simulation Execution*

The start phase is followed by the execution phase in which the simulation models are synchronized and relevant events describing the material and information flow between simulation models are exchanged.

The basic modeling of material and information flow between two coupled models M_i and M_{i+1} is described in section III. Both models send/receive interactions and object state changes. This data traffic is essential for modeling the synchronized transfer of parts to a buffer with limited capacity. Moreover, the observer federate listens to this traffic, too. The observer federate collects data needed to compile dedicated statistics for single simulation models as well as for the federation. Traffic data is also be used to generate feedback information, e.g. on the simulation's progress, for the user during simulation execution.

The execution phase finishes automatically once the simulation clock reaches the preset end time in all federates. All simulation model federates resign the federation and start their own post-run preparations to output data.

V. DATA ACQUISITION

The observer federate listens to data traffic between the model federates. Measures of the complex simulation model can be gathered from the collected data and prepared for the user. The observer federate is generated for every federation based on the FED-file. All information needed to create the observer federate is drawn from the FED file.

The observer federate subscribes dedicated interactions between other federates. These interactions are sent automatically by federates. Most subscribed interactions are part-specific. An interaction is created when the state of a part is changed in the content of outcomes. Some interactions are specific to the simulation model, e.g. the number of parts produced per day. Table 1 shows some of the interactions that are used to communicate evaluation related data.

TABLE I. EXAMPLES OF INTERACTIONS

Interaction	Description
Create Part	Creating time of a single part
Daily Statistic	Numbers of demand and simulated outputs
Delay Part	Start and arriving time for sending a single part from one model to another
Finish Part	Finish time for a single part in the production process
Load Part	Loading time on a carrier for a single part
Start Part	Starting time for a single part in the production process
Unload Part	Unloading time from a carrier for a single part

Sending these interactions requires additional software code within the simulation models. In our example these enhancements are automatically inserted into the source code when the simulation models run in the distributed mode. No additional user actions are necessary. In this case the architecture of the used generic simulation models enables this very efficient approach.

VI. ARCHITECTURES

The main components of our approach of distributed HLA simulations in industrial applications are:

- the HLA Run Time Interface (RTI),
- simulation models,
- the Observer model, and
- a support system.

The relationships between all components are varying during the simulation. Relationships during the execution phase are depicted in Figure 4. Figure 3 shows relations during the starting phase.

The integrated support system must perform the following important functions:

- provide a graphical user interface focused on configuring simulation models and presenting the output data,

- perform activities during the configuration phase such as checking model relationship validity and creating the global configuration file and
- automatic execute the monitoring phase and generate the simulation outcome.

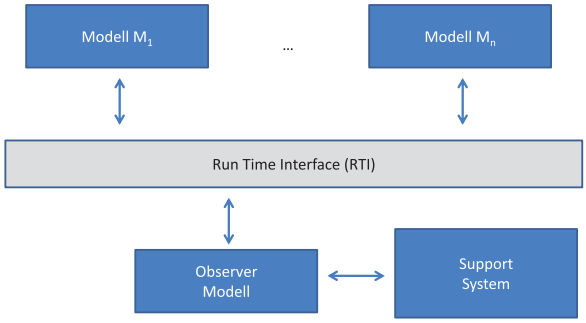


Figure 4. Distributed HLA simulation components

Our support system consists of two main components, a **graphical user interface** (GUI) for user interaction and a **commander** that controls and execute all the operations described for distributed simulation. Both components are implemented in C# using Microsoft .NET Framework. The used basic software for HLA is the Open Source HLA implementation from CERTI [11]. Additionally a wrapper library has been developed that wraps HLA-functions and callbacks for the usage within the C#-based user interface.

The support system architecture meets factory planners and operators' requirements for the execution of distributed simulations. Users interact with the user interface only to prepare, run and analyze distributed simulations. Users only insert that data that cannot be derived from existing data sources within the simulation models.

The commander manages all necessary resources and components. The resources simulation models are time-persistent, whereas the FED file and configuration files are temporary. The files are generated during the distributed simulation and deleted upon its completion. The observer component is also generated and exists during the monitoring phase only. Commander components are shown in Figure 5.

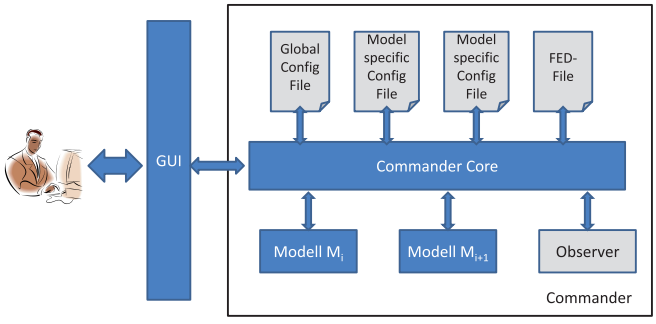


Figure 5. Commander components

VII. OUTCOMES

Two different kinds of outcomes are presented to the user: measures and animations. **Measures** characterize the state of the entire complex system. Data collected during execution phase by the observer federate is the source of appropriate measures. Based on the raw data, the commander aggregates, prepares and calculates statistical parameters for measures. **Animation** depicts part flows and states of buffers on an abstract level over time.

Typical measures for complex production systems are:

- Delays in a receiver component due missing incoming parts from deliver factory components,
- Waiting times of parts in a deliver component due to unavailability of receiving buffers,
- Buffer content progress,
- Overall production in relation tofor required demands and simulated output,
- Overall system cycle times and
- Part history.

The added value of the measures is the quantification of the dependencies between coupled models. Buffer content and waiting time measures enable the user to determine design data for coupling features between single factory sections. A typical task for factory planners is the design of transport capacities and buffer sizes between different sections of a factory. The calculated measures aid the identification of bottlenecks before the factory is implemented. Experiments with the complex simulation model can serve as a basis for the validation of new proposed improvements of the coupling features.

The results presented here are representative values for a production system consisting of four production components. Each component has its own simulation model. Components 1 to 3 (models M1, M2 and M3) deliver parts to component 4 (model M4). There are buffers with specific limited capacities for each material providing component inside of receiving components. Figure 6. illustrates the material flow between the components.

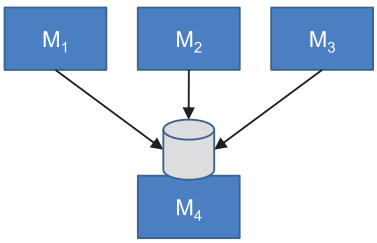


Figure 6. Example of Material flow

TABLE II. provides an example of measured results from model M4 for waiting times. Models M1, M2 and M3 deliver parts to model M4. The production in model M4 must stop when parts needed from provider models are unavailable.

TABLE II. DELAY TIMES FOR MODEL M4 CAUSED BY MISSING INCOMING PARTS FROM PROVIDER MODELS

Delays caused by missing incoming submodels			
Model	Provider Models		
M4	M1	M2	M3
# of delays	0	10	4
Minimum delay	0s	30m	4m
Average delay	0s	1h 45m	1h9m
aximum delay	0s	7h 34m	2h23m
Total delay	0s	17h 30m	4h34m

Another frequently used measure is buffer content progress over time. Such graphs facilitate the identification of bottlenecks and possible causes. One such graph is presented in Figure 7.

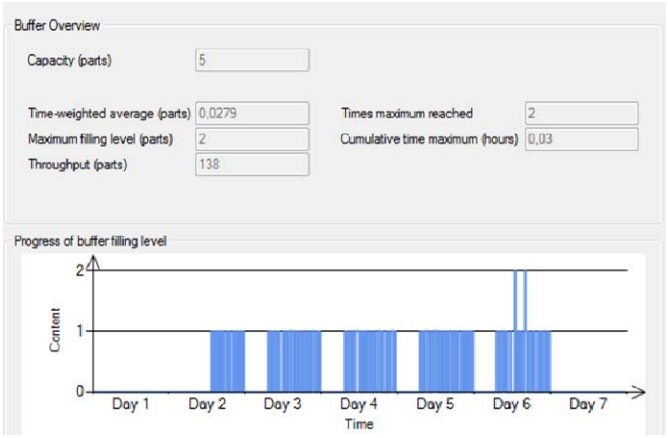


Figure 7. Content of the buffer between model M1 and M4

Figure 8. shows an example of the comparison of the demand and the simulated production output for one sub model. The sub model depends on other sub models that deliver parts. Production output is contingent by the provider models and can only be calculated in a complex model.

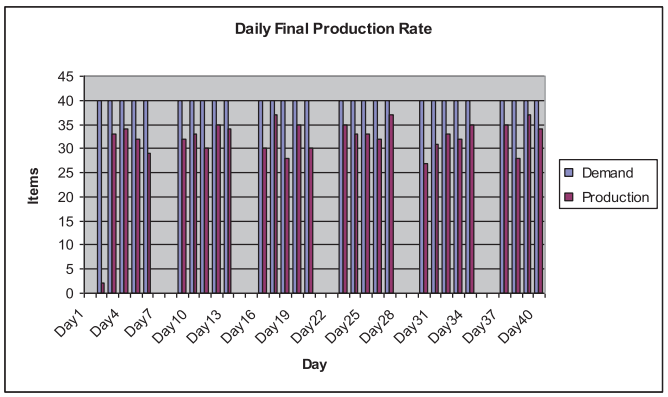


Figure 8. Daily demand and simulated production output for a sub model

An additional advantage is the capability of modeling measures for the overall process. One such measure is the total cycle time for a complex product from the fabrication of the first subpart through completed assembly of the entire product. This measure can only be calculated when the related simulation models are coupled. Figure 9. presents an example of a total cycle time measure.

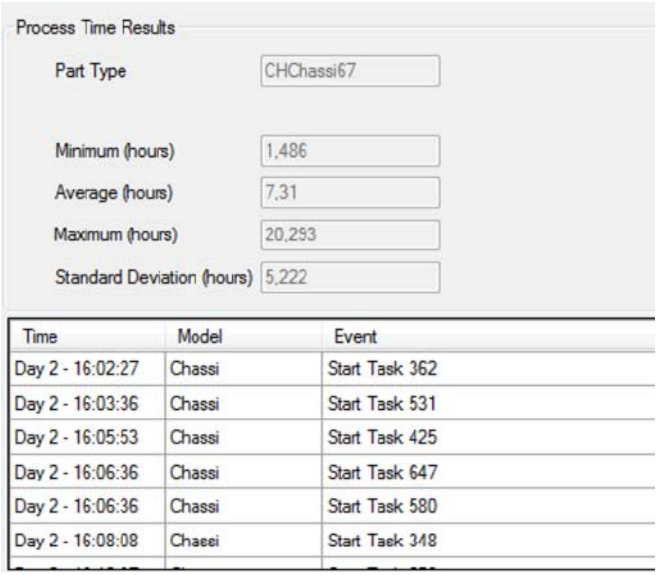


Figure 9. Example for total cycle time measures

Generally, an individual visualization model was created for each of the monolithic simulation models. These models were inappropriate for the visualization of the distributed simulation since only the material flow *between* the individual models were of interest in this use case (i.e., the internal actions *within* a model were not to be visualized). Therefore, a new visualization including statistic features of the interesting outputs had to be created.

VIII. CONCLUSION

Significant conclusions about the performance of a complex system cannot be unconditionally drawn from the aggregation of the performances of single components modeled without the analysis of the relationships between the single components. The reuse of legacy simulation models in a federation based on the distributed simulation approach should be preferred compared to the creation a new monolithic model of the entire complex system from scratch.

Distributed simulation models have to be prepared and used by simulation experts, which is still a very time consuming process. Since simulations have to be used not only by IT and simulation experts but also by factory planners and operators those efforts and complexities have to be lowered. This can be archived by offering some kind of support system for distributed simulations. The prospective users may concentrate only on the selection of models to integrate, the description of valid model relationships and interpretation of the simulation outcomes. Other necessary activities should be hidden from the user and executed automatically by the support system.

The support system and the approaches developed are transferrable to other applications e.g. logistics system coupling and supply chain modeling.

Future research and development will focus on:

- reduction of simulation execution runtimes for distributed material flow models,
- visualization of single models or comprehensive models with different levels of abstraction and
- development of approaches to design distributed simulation experiments intended to optimize complex production systems.

REFERENCES

[1] S. AbouRizk, Y. Mohamed, H. Taghaddos, F. Saba and S. Hague, "Developing Complex Distributed Simulation for Industrial Plant Construction using High Level Architecture" WSC2010

[2] C A. Boer, A de Bruin and A. Verbraeck, "Distributed Simulation in Industry – A Survey, Part 3 – The HLA Standard in Industry, in Proceedings of the 2008 Winter Simulation Conference, S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds., pp 1094-1102.

[3] C. A. Boer,"Distributed Simulation in Industry", PhD Thesis Erasmus University Rotterdam, 2005.

[4] M. Rabe, and F.-W. Jaekel. "The MISSION project – demonstration of distributed supply chain simulation", in Enterprise inter- and intra-organizational integration, 235-242. Boston, Dordrecht, London: Kluwer. 2004

[5] S. Strassburger, G. Schmidgall, and S. Haasis. "Distributed Manufacturing Simulation as an Enabling Technology for the Digital Factory", in Journal of Advanced Manufacturing Systems (JAMS), 2003,. 2(1), pp.111-126.

[6] P. Lendermann, "About the Need for Distributed Simulation Technology for the Resolution of Real-World Manufacturing and Logistics Problems", in Proceedings of the 2006 Winter Simulation Conference, L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds., pp. 1119-1128

[7] SISO. 2010. Standard for Commercial-off-the-shelf Simulation Package Interoperability Reference Models. SISO-STD-006-2010. Available via <http://www.sisostds.org> [accessed Janury 2011]

[8] S. Strassburger, T. Schulze, and R. Fujimoto, "Future Trends in Distributed Simulation and Distributed Virtual Environments" Peer Study Final Report. Version 1.0. Ilmenau, Magdeburg, Atlanta – January 17, 2008.

[9] S. Strassburger, T. Schulze, and M. Lemessi, „Applying CPSI Reference Models for Factory Planing", In Proceedings of the 2007 Winter Simulation Conference, S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds. pp 603-609.

[10] M.Raab, T. Schulze, and S. Strassburger, „Management of HLA-based Distributed Legac SLX-Models", In Proceedings of the 2008 Winter Simulation Conference, S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds., pp. 1086-1093.

[11] CERTI 2011. Open Source HLA RTI. Available via http://savannah.nongnu.org/projects/certi/ [accessd January 2011]

Approximation of dispatching rules in manufacturing control using artificial neural networks

Sören Bergmann
Ilmenau University of Technology
School of Economic Sciences
Department for Industrial Information Systems
Ilmenau, Germany
soeren.bergmann@tu-ilmenau.de

Sören Stelzer
Ilmenau University of Technology
School of Economic Sciences
Department for Industrial Information Systems
Ilmenau, Germany
soeren.stelzer@tu-ilmenau.de

Abstract - **Automatic generation of simulation models has been a recurring topic in scientific papers for years. A common problem of all known model generation approaches is the generation of dynamic behavior, e.g. buffering or control strategies. This paper introduces a novel methodology for generation of dynamic behavior, based on artificial neural networks, which is usable directly in the simulation. We also test the approach in a manageable scenario; all results are illustrated via small simulation experiments.**

Keywords - ***Dispatching Rules, Artificial neural networks, Simulation model generation, function approximation***

I. INTRODUCTION

Simulation, especially discrete-event simulation, is used within many different disciplines and application areas. In the area of production and logistics, simulation is a well accepted tool for the planning, evaluation and monitoring of relevant processes. It is essential to adequately model reality in a simulation model in a way that allows sufficiently exact predictions about the real system. The quality of the simulation based predictions directly depends on the quality of the model. The modeling process for achieving high quality models which are verified and validated is time consuming and usually requires a simulation expert.

In this context Fowler and Rose [1] have discussed future challenges for modeling and simulation of complex production systems. Among others, they identified the reduction of the time and effort for simulation studies as future research areas. Given these challenges, approaches to automatically generate simulation models seem to be very appealing. The promise of such approaches is that they, if successful, can reduce the amount of time needed to create a simulation model as well as the expertise needed for creating and conducting simulations and if applicable also to improve the quality of the model [2].

A common problem of all known model generation approaches is the generation of dynamic behavior. Information about behavior, e.g. buffering or control strategies, is often not stored in the data sources and IT systems used for model generation or not explicitly known at all (often done on an ad-hoc basis). This is attributed to the fact that the behavior is not relevant for the most operational use cases and/or the behavior is so complex that only algorithmic descriptions capture it correctly [2] [3].

First approaches for determining control strategies, such as dispatching rules, were examined in recent years.

For instance Selke, Gyger and Reinhard investigated pattern recognition (data mining) techniques to identify control strategies, based on historic production data (production data acquisition - PDA) [3], [4]. A similar approach is used by Xiaonali and Olofson resulting in decision trees as output [5]. The applicability of all these approaches is very limited in the context of automatic model generation or possible only under restrictive conditions.

The idea of our approach is not to exactly identify control strategies, especially scheduling strategies, but to approximate them through artificial neural networks.

The primary goals are speed up the modeling process and raise accuracy of the simulation model, not main goal is to speed up the single simulation run. But a speed up of the simulation runs can be a nice side effect.

The remainder of the paper is organized as follows. In Section 2, we discuss dispatching rules in the manufacturing control, what parameters are relevant and why we can interpret dispatching rules as a function. In Section 3 we briefly review some related works, how artificial neural networks are used in manufacturing control.

In Section 4 we show how artificial neural networks can be used for the approximation of dispatching rules and we evaluate the performance of the approach. Finally, Section 5 contains some concluding remarks and future research directions.

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

II. DISPATCHING RULES IN MANUFACTURING CONTROL

The different decisions that have to be made in production planning and control (PPC) fall into four main strategy clusters:

- strategies of job release
- strategies of sequencing
- strategies of lot size
- strategies of resource scheduling [3], [6]

Dispatching rules are used to decide the priorities for fulfilling orders and can be used for all 4 clusters. In this paper we focus on dispatching rules for scheduling jobs. In practice dispatching rules are popular. Their popularity can be attributed to their relatively easy implementation in practical settings and their minimal computational effort and information requirements [3] [7].

A large number of dispatching rules for scheduling jobs can be found in the literature. These rules are well studied and are known to perform well for a given set of objectives and constraints.

Dispatching rules can be distinguished by the information they need as input. There are rules which only required local information (e.g., about the current buffer contents in front of a machine etc.) and there are rules that use global information (e.g., about all buffers etc. in the manufacturing system).

Based on the type of information required by the rule different classes of rules can be identified (Table 1). The result of a rule is a priority of a job; the actual processing order is then obtained by sorting these priorities [6], [8].

The single rules are in practice often combined, two variants are possible: additive or multiplicative combination. Additive combination means that a second rule is used when two or more jobs have the same priority, e.g. if SPT is used and 2 jobs have the same processing time, it is necessary to decide which has the higher priority. In such cases FIFO is the last used rule, because FIFO is always well-defined. Multiplicative combination occurs when two or more rules are used in parallel and all individual result priorities will be weighted and charged.

Table 1: classes of dispatching rules

Characteristic	Example for a rule
Entry-time oriented	FIFO (First In First Out) [in system →global or in buffer → local]
Process-time oriented	SPT (Shortest Processing Time),
Setup-time oriented	Minimum Setup Time
Date oriented	EDD (Earliest Due Date), ERD (Earliest Release Date)
Customer priority based	Highest Customer priority
Value/Cost based	Maximum contribution margin, maximum previously generated costs
Combination	WINQ (total work-content of jobs in the queue of the next operation of a job), SL (Slack Time), COVERT (Cost over Time)

Each characteristic is defined by a finite number of attributes of the job. Possible attributes are:

- Arrival time (in buffer, in whole system)
- Process Time (next machine, all remaining machines)
- Setup time
- Due date (or Delta to the due date)
- Priority (-flag)
- Buffer-Queue length (this, all, next Buffer)

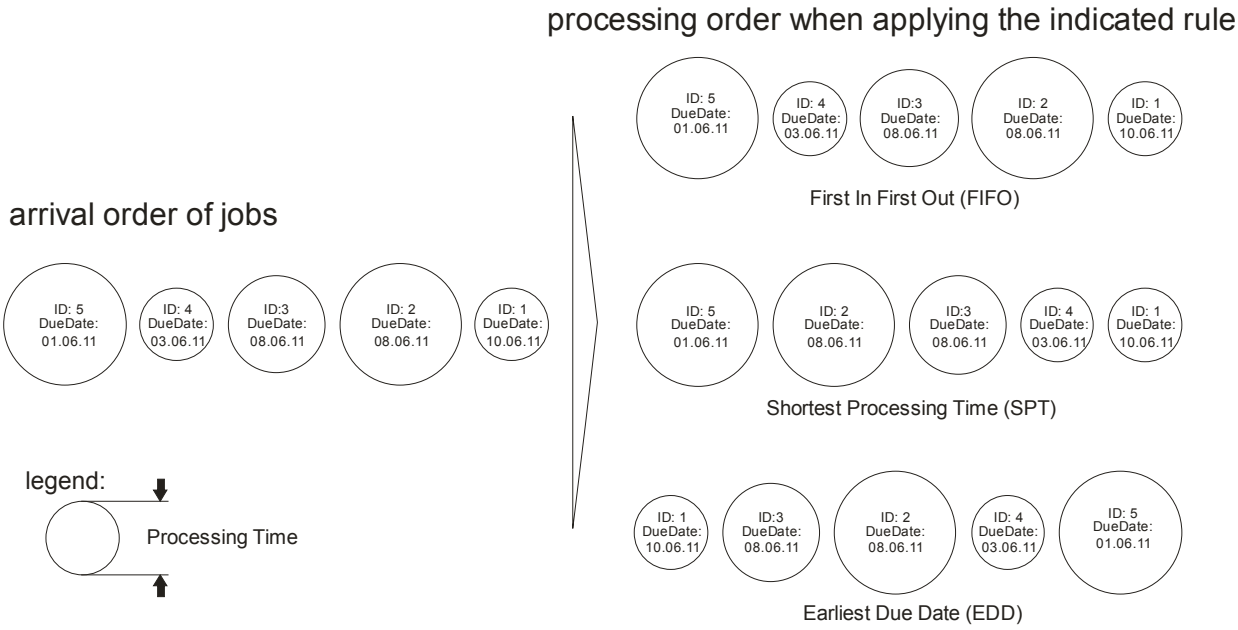


Figure 1: Use of dispatching rules (e.g.: FIFO, SPT and EDD)

- Processing costs (on this, all, previous or following processing machines)
- Setup costs
- Storage costs
- Internal retail price
- Etc.

Additional parameters can be calculated from the attributes, e.g. Slack (Due Date - Process Time on all remaining machines), contribution margin (Internal retail price – all costs).

All rules (both simple and multiplicative rules) can be interpreted as functions of which the arguments (input) are job attributes and the return value is the assigned job priority. All attributes have a special weight, unused attributes are weighted with zero.

(1) Decision rule → Priority function

$$P_i = \sum(g_j * a_{ij})$$

where: P_i: priority of job i

a_{ji}: attribute j of job i

g_j: weight for attribute j

In this paper for all examples we use three typical rules: FIFO, SPT and EDD (Fig.1), the attributes which are relevant for these rules are arrival time at the current buffer (t_{arr}), processing time on the current machine (t_{pt}), and the due date (t_{dd}) of each job in the buffer queue.

These example rules can be defined as functions as follows.

(2) FIFO: $P_i = \frac{1}{t_{arr\,i}}$

$$\rightarrow P_i = \left(1 * \frac{1}{t_{arr\,i}}\right) + \left(0 * t_{pt\,i}\right) + \left(0 * \frac{1}{t_{dd\,i}}\right)$$

(3) SPT: $P_i = \frac{1}{t_{dd\,i}}$

$$\rightarrow P_i = \left(0 * \frac{1}{t_{arr\,i}}\right) + \left(1 * t_{pt\,i}\right) + \left(0 * \frac{1}{t_{dd\,i}}\right)$$

(4) EDD: $P_i = \frac{1}{t_{dd\,i}}$

$$\rightarrow P_i = \left(0 * \frac{1}{t_{arr\,i}}\right) + (0 * t_{pt\,i}) + \left(1 * \frac{1}{t_{dd\,i}}\right)$$

(5) Composition of simple rules

$$P_i = \left(0 * \frac{1}{t_{arr\,i}}\right) + (0.4 * t_{pt\,i}) + \left(0.6 * \frac{1}{t_{dd\,i}}\right)$$

From these functions, a priority for each job is obtained. The job with the maximum priority is the job which will be processed next.

In section 3 we show how artificial neural networks can be used for the approximation of these functions.

III. ARTIFICIAL NEURAL NETWORKS

Artificial neural networks (ANN) are universal and highly flexible function approximators and often used in domains where the behavior of the analyzed system cannot be described by a mathematical model [9]. ANNs can be understood as a non-linear mapping structure, which is inspired by the human brain. The basic components of an ANN are simple computation entities, analogue to brain-structure called neurons, which are linked to other neurons by weighted connections. To simplify the complexity of biological neurons, ANNs often use very simple neuron models, where every neuron has an own activation level, which depends on the activation of connected neurons [9], [10]. To calculate the activation every neuron has to summarize their inputs, considering the corresponding weight. To achieve a non-linear behavior the calculated sum is transformed by an activation-function.

That way, the input-output mapping of an ANN is fully described by the weights of the connections and the used neuron-model. To create an ANN, which represents the desired mapping, they have to be trained. A common way to train ANN is supervised learning, which requires a set of examples of the desired mapping or function. These samples, also called trainings data, are presented to the ANN, which adapts its connection weights, to improve its output performance. Almost every learning algorithm proceeds similar to the following steps: present the input to the neural network and determine the error between its output and the expected output on the selected training data. Next, adapt the connection weights to reduce the measured output error.

There are a whole set of learning algorithms for ANN, which addresses various network configuration or problem structures. The most common algorithm for training is the Back-Propagation (B-Prop) algorithm, also called Error-Back-Propagation. It addresses Feed-Forward Networks, which means that there is a directed information processing graph from input neurons to output neurons. So the networks output error can be propagated backwards through the network. Analytically, the B-Prop algorithm is a gradient based optimization algorithm to minimize the output error for a set of learning data [9].

The mapping-complexity of ANN grows with the count of the neurons and the connection rate. To determine the complexity, several network structures have been studied. The Multi-Layer-Perceptron (MLP) is a class of networks, which consists of a set of neuron layers that are uni-directional connected to each other.

Over the years, the regard and appreciation of ANNs in research had its ups and downs, but in some domains they are very attractive, especially in robotics, pattern-recognition, classification or function-approximation. In other domains, like manufacturing or logistics control, the approaches based on ANNs are sparsely represented. We have identified some relevant publications describing the use of ANN to solve

problems in manufacturing and logistics. On job-scheduling problems, ANNs are well investigated. However, they are very rarely used in conjunction with discrete event simulation. The most of these works try to transform the given Job-scheduling problem in a designed network structure, which solves the problem, by adapting certain network parameter, e.g. connection weights. Jain and Meeran [11] do apply Hopfield-Networks which are trained by a modified Error-Backpropagation Algorithm to find near-optimal solutions. After learning or rather adapting, the solution has to be extracted by analyzing the resulting ANN structure. Another approach for solving job-scheduling is introduced by Anilkumar and Tanprasert [12], where ANN are evolved that are mapping the given job-list to a priority value. To calculate the mapping the job-attributes and queue-information where set as input. Afterwards the job-schedule can be formed by sorting the jobs by their priorities.

This kind of approaches tries to find optimal job-schedules, but ANNs can also be used to learn the behavior of systems or sub-systems. A key aspect of modeling using discrete event simulation is to break down a system to basic elements, which have to be modeled. To model these simple components of larger systems ANN could also be used. In the approach of Markwardt et. al. [13], they try to model the major elements of a material flow system with ANN, by learning the input-output behavior. Dynamic behavior, initiated by decision rules, is not regarded in their works.

IV. Approximation of Dispatching Rules using ANN

In this section we show how ANNs can be used for approximation of unknown or not well described dispatching rules in the modeling process.

To be able to do this, the ANN firstly needs to be trained. In a practical setting, we suggest to use data which is obtained from the real productions system for the training. In the research work presented here, we have used a simulation model emulating the real system. In this model, the dispatching rules are modeled correctly by using the original control strategies of the simulator.

At first we explain how to obtain and encode the empirical input data into a form usable for the ANNs. Afterwards we show how the network output can be transformed into dispatching decisions and discuss the learning process of the ANN. At the end of this section we present our results.

Our test scenario is a simple single server model. Of this model, two versions exist: one for generating training data for the ANN, and one using the ANN as approximator for the dispatching decisions. In the models, an item source periodically generates job orders of a given job-mix (Table 2).

If the machine is blocked the waiting jobs are placed into a buffer in front of this machine. Every time the machine finishes a job, a new job is selected from the buffer, considering the selected dispatching rule or its approximation by the ANN. In training mode, in this situation a snapshot of the buffer (including all job information) and the selected job is taken and inserted into the training data.

Table 2: Job Parameters

Job type	processing time (PT) [in minutes]	Frequency [in %]	Due date
A	2:00	20	$= t_{current} + (\text{PT} * \text{uniform}[1.5 \dots 2.5])$
B	3:00	20	
C	4:00	20	
D	5:00	20	
E	2:30	20	

To reflect a well loaded system, we setup the job-creation routine to achieve a system load of 70%. The whole simulation was limited to 40h. To consider the chosen network structure, the buffer size has to be set to 5.

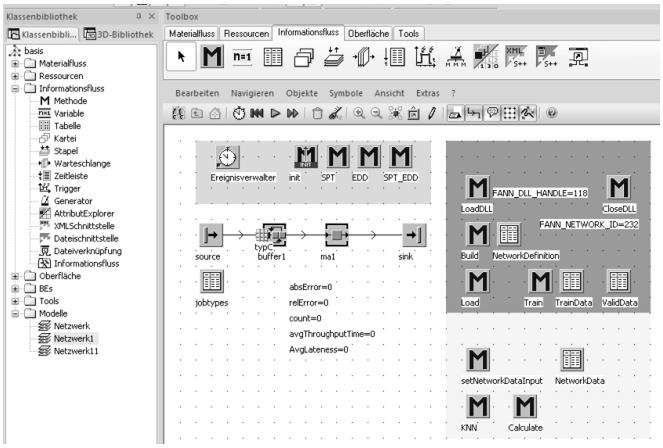


Figure 1: Plant Simulation test scenario

To use an ANN to approximate the empirically collected decision rule information, we had to transform the input data obtained in the snapshots. We assume that the same input data used here could also be obtained from a real material flow system. In our scenario the size of an input vector scales with the buffer size and the collected job attributes. The constant buffer size and the chosen job-attributes (Table 2) lead to network configuration with 5, 10 and 15 input neurons. To improve the training of the ANN we normalize the input vector to the interval (-1; 1). To perform this normalization, the min-/max values of the job-attributes have to be determined.

(6) Processing Time (PT)

$$normalize(PT_i) = \frac{PT_i * 2}{max(PT)} - 1$$

Additionally, for attributes which contains date related information, like EDD, an offset has to be defined.

(7) Due Date (DD)

$$normalize(DD_i) = \frac{(DD_i - t_{current}) * 2}{max(\Delta DD_i)} - 1$$

where: ΔDD is the remaining time till due data

In the next step, we had to choose a well suited output coding. In our studies, we choose a binary coding. So the output vector had to be set to a value of 0 or 1, where values of 1 for job i means that it will be processed next. In the test-data creation process, we ensure that there are no concurrencies. If a decision rule results in an ambiguous situation, we choose FIFO to solve it.

(8) Output-Coding:

$$o_i = \begin{cases} 0, & if\ job\ i\ is\ processed\ next \\ 1, & otherwise \end{cases}$$

After the definition of the input- and output-coding, the network parameters and topology had to be determined. In our experimentation we used a small MLP with one hidden-layer, which was varied in size.

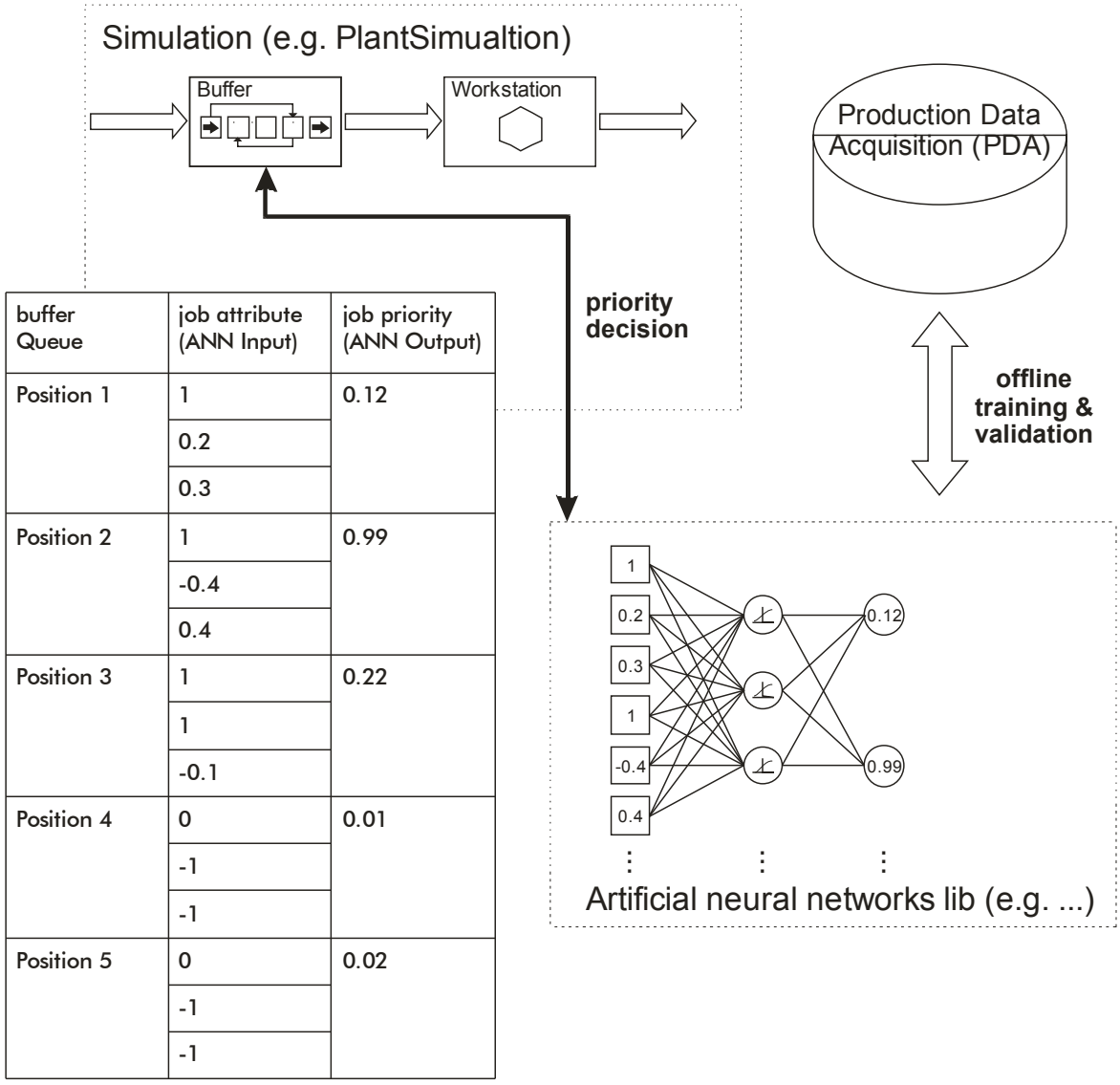
For activation we applied the logistic-function, which is a non-linear sigmoid function. The usage of a non-linear activation function is crucial for the mapping complexity of the ANN.

To reflect a given input-output mapping, the ANN has to be trained. We were using the well studied B-Prop algorithm. To determine the train-error, which means the delta between network output o and expected output t from the train-samples, we were using the Mean-Square-Error (MSE) function. In literature, the MSE in combination with the B-Prop algorithm MSE function is achieving positive results.

(9) MSE:

$$E = \sum_{i=0}^{n} 0.5 * (t_i - o_i)^2$$

where: n is the output dimension



buffer Queue	job attribute (ANN Input)	job priority (ANN Output)
Position 1	1	0.12
	0.2	
	0.3	
Position 2	1	0.99
	-0.4	
	0.4	
Position 3	1	0.22
	1	
	-0.1	
Position 4	0	0.01
	-1	
	-1	
Position 5	0	0.02
	-1	
	-1	

Figure 2: Test-Environment used for Validation

To prove the basic function of our approach; we designed two simple test-cases, where the ANN had to learn the SPT dispatching rule offline. The network input was composed of the processing times in first case. In the second case, information about the buffer state was additionally given. From these cases we derived several experiments, where we varied the learning parameter and topology of the ANN and composition of the training data.

The results of our experiments were useful for understanding the sensitivity of learning and network parameters, so we learned that the SPT decision rule with a binary output coding could not be approximated by a simple MLP without any hidden layer. Based on these experiences for further tests we choose MLPs with one hidden layer, which was varied from 5 to 20 Neurons. In addition to adapting the topology of the neural network, we had to tune the learning parameters, like the learning rate or the count of parameter iterations (epochs).

Table 3: Learning and network parameters

Parameter	min	max	best
Learning rate	0.05	0.4	0.1
Epoches	50	10000	200
Hidden Layers	0	5	1
First Hidden Layer size	2	20	10

To avoid an overfitting [10] effect of the ANN the training-data was split in learning-data and validation-data, so the ANN could be trained by using a subset of samples of the desired mapping an validated by another set of samples. The learning- and validation error could be used to terminate the learning process, in our case the learning of the ANN was canceled in two cases. In case 1, the mapping error of the ANN falls under the defined error limit. In the second case the maximum of learning epochs was achieved.

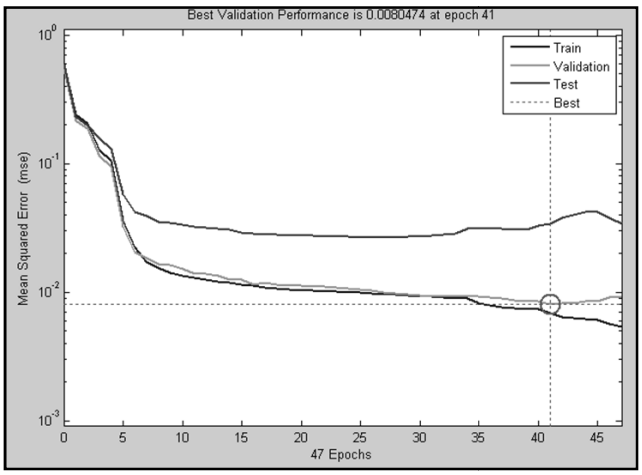


Figure 3: Train and Validation error over learning-time

As we have the desired mapping in form of trainings samples, we use offline-learning. That implies, that the adaption of the connection weights is applied after presenting the whole set of learning data. The learning parameters of the best performing experiments were used to setup the following investigations.

For all tests Plant Simulation was uses used as the simulation environment. The implementation of the ANN was realized by an external library linked to the simulator. An overview of the components and their information flow is shown in fig. 2. The offline-learning process additionally was validated in the Matlab Neural Network Toolbox and the JNNS.

V. EXPERIMENTS

After the experiments to prove the capabilities of our approach and determine the learning parameters, we expand our investigations to the EDD and a composed decision rule. To comparison we also took the FIFO decision rule.

Our experiments targeted to check how well the artificial neural networks maps the dispatching rules and how a system behaves, if an ANN is dispatching instead of the original control strategies of the simulating environment. To compare the system, we measured the average cycle time, average lateness and schedule variance for both cases under same conditions, the results are show below (Table 5). A more detailed comparison is shown for EDD (fig. 3), SPT (fig. 4) and FIFO (fig. 5). Every decision was parsed and any deviations between the behavior of the ANN and the behavior of the rule were determined (Table 6).

Table 4: Comparison of the modeled systems by ANN and original control strategy (ocs)

Decision rule	Avg. cycle time in [min]	Avg. lateness in [min]	schedule variance in [min]
SPT (ANN input=5)	10:13	9:03	-3:16
SPT (ANN input=10)	10:08	9:07	-3:10
SPT (ANN input=15)	10:11	8:49	-3:13
SPT (ocs)	10:05	8:47	-3:07
EDD (ANN input=15)	11:20	7:37	-4:12
EDD (ocs)	11:15	7:20	-4:08
FIFO (ANN input=15)	11:36	7:46	-4:26
FIFO (ocs)	11:36	7:46	-4:26
SPT+EDD (ANN input=15)	10:11	8:39	-3:13
SPT+EDD (ocs)	10:05	8:45	-3:07

Table 5: Experimentation results, simulation time=40h

Experiment ANN Training	MSE (avg.)	Error [abs]	Error [rel]
SPT (input=5)	0.0029	26	6,52
SPT (input=10)	0.0061	18	4,51
SPT (input=15)	0.0028	25	6,27
EDD (input=15)	0.0024	33	8,15
FIFO (input=15)	0.0001	0	0
SPT+EDD (input=15)	0.0061	34	8,52

In direct comparison of the simulation environment in-built decision rules and the trained ANN we never determined an error rate higher than 8.5%. Both modeling methods were compared in the test environment using validation data. To notice that the validation data were never trained, we can assume that the ANN has learned the respective function representing a given decision rule. The error rate over the data used for training the ANN is much lower (table 6).

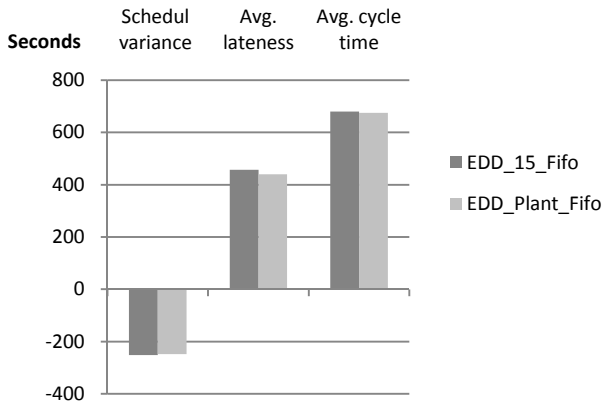


Figure 4: evaluation rule SPT vs. ANN SPT

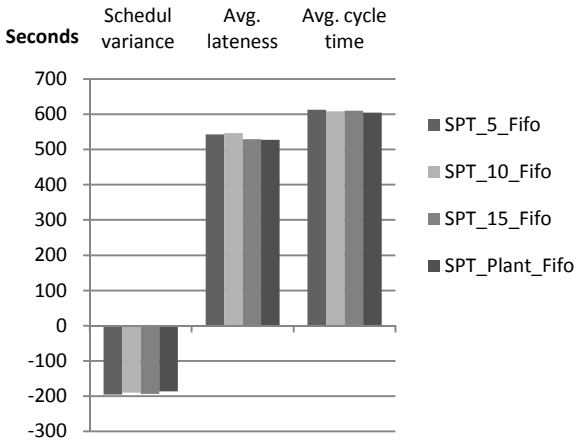


Figure 5: evaluation rule SPT vs. ANN SPT

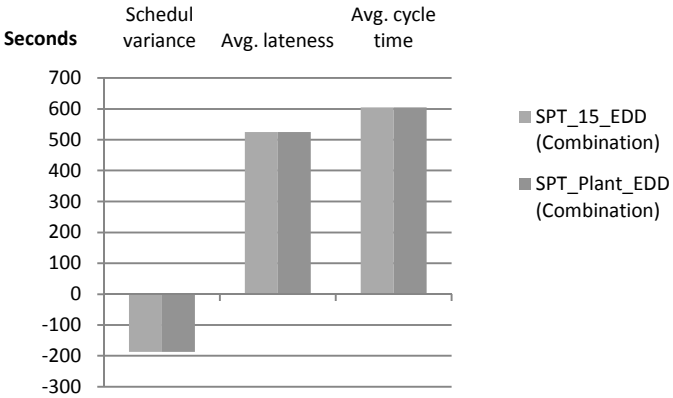


Figure 6: evaluation rule SPT + EDD and ANN SPT + EDD

The impact of the behavior of our test scenario is negligible, if decision rules are modeled by ANNs as shown in our results for EDD (fig. 4), SPT (fig. 5) and SPT (fig. 6).

In our experiments to model dynamic system behavior, like dispatching rules, could approximated by ANN. We could also show that a minimal mapping error exists, but the impact of that error is negligible.

VI. SUMMERY AND DISCUSSION

We have introduced a new methodology for emulating dynamic behavior, based on artificial neural networks, which is usable directly in the simulation. We show, in a manageable simulation model, that artificial neural networks are usable for approximation of decision rules. In our example Plant Simulation was used as simulator, but the integration of ANN is doable in every DES which supports external library interfaces.

This paper represents a first step, to effectively use ANN in the context of automatic simulation model generation. Our approaches represent a solution to the common problem of existing model generation approaches which often fail to reproduce the dynamic behavior of the real system.

It must be further examined how well this approach performs for more complex scenarios, e.g. in situations with multiple machines, various buffer sizes and with a larger set of consolidated information (more local information and also global information). For growing number of input neurons (number buffer places * number of used information) it will be interesting to conduct performance studies.

On the other hand it will be interesting to see, how the results change when the parameters of neural networks are changed. In detail especially different encodings of input and output data, various network topology, e.g. simple more hidden layers in the MLP or recurrent networks like Jordan or Elman Networks, could be examined. Depending on the network topology various learning methods for artificial neural networks need to be investigated on their impact. It is also possible to evaluate techniques such as neuroevolution or R_{max}.

Furthermore, it is interesting to investigate how the use of ANN affect the speed and runtime behavior of single simulation runs, first simple tests indicate that a speed up is reachable under certain conditions.

VII. REFERENCES

[1] Fowler, J. W., Rose, O. „Grand Challenges in Modeling and Simulation of Complex Manufacturing Systems“. SIMULATION: The Society for Modeling and Simulation International, 80(9): 469–476, September 2004.

[2] Bergmann, S., Straßburger, S. „Challenges for the Automatic Generation of Simulation Models for Production Systems“. In: Proceedings of the 2010 Summer Simulation Multiconference, 11-15 July 2010, Ottawa, Canada, pp.545-529.

[3] Reinhart, G., Selke, C. „ Information Becomes Knowledge – Automatic Building of Simulation Models“. In: Proceedings of the 35th CIRP International Seminar on Manufacturing Systems, 13–15 May 2002, Seoul, Korea.

[4] Reinhart, G., Gyger, T.; "Identification of implicit strategies in production control," Industrial Engineering and Engineering Management, 2008. IEEM 2008. IEEE International Conference, pp.302-306, 8-11 Dec. 2008.

[5] Xiaonan, L., Olafsson, S. "Discovering dispatching rules using data mining“. Journal of Scheduling, vol. 8, no. 6, pp. 515-527, 2005.

[6] Milberg, J., Selke, C. "Automatic generation of simulation models for production planning and control," In: Annals of the German Academic Society for Production Engineering, vol. 10, no. 1, pp. 115-118, 2003.

[7] Pinedo, M. “Scheduling: Theory, Algorithms and Systems”, Prentice Hall, Englewood Cliffs, NJ 2002.

[8] Mertens P. (2000) „Integrierte Inforationsverarbeitung 1 – Administrations- und Dispositionssysteme in der Industrie“ edition 12. Gabler, Wiesbaden 2000.

[9] Zell, A. „Simulation Neuronaler Netze.“, Oldenbourg Verlag, 1997

[10] Bishop, C. „ Neural Networks for Pattern Recognition.”, Oxford UP, 1995

[11] Jain, A. S., Meeran S., “Deterministic job-shop scheduling: past, present and future”. European Journal of Operational Research 113, pp. 390–434, 1999

[12] Anilkumar K. G., Tanprasert T., “Neural Network Based Priority Assignment for Job Scheduler”, AU Journal of Technology Vol. 9 Nr. 3, 2006

[13] Markwardt, U., Schulz F.; Marquardt H.-G., “Modelling material handling systems by means of artificial neural networks”, In: Russell Meller U.A. (Hrsg.): Progress in Material Handling Research, 2004

A Virtual Time System for OpenVZ-Based Network Emulations

Yuhao Zheng
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
zheng7@illinois.edu

David M. Nicol
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Urbana, Illinois, USA
dmnicol@illinois.edu

***Abstract*—Simulation and emulation are commonly used to study the behavior of communication networks, owing to the cost and complexity of exploring new ideas on actual networks. Emulations executing real code have high *functional* fidelity, but may not have high *temporal* fidelity because virtual machines usually use their host's clock. A host serializes the execution of multiple virtual machines, and time-stamps on their interactions reflect this serialization. In this paper we improve temporal fidelity of the OS level virtualization system OpenVZ by giving each virtual machine its own virtual clock. The key idea is to slightly modify the OpenVZ and OpenVZ schedulers so as to measure the time used by virtual machines in computation (as the basis for virtual execution time) and have Linux return virtual times to virtual machines, but ordinary wall clock time to other processes. Our system simulates the functional and temporal behavior of the communication network between emulated processes, and controls advancement of virtual time throughout the system. We evaluate our system against a baseline of actual wireless network measurements, and observe high temporal accuracy. Moreover, we show that the implementation overhead of our system is as low as 3%. Our results show that it is possible to have a network simulator driven by real workloads that gives its emulated hosts temporal accuracy.**

Keywords—network emulation, virtual time, virtual machines

I. INTRODUCTION

The research community has developed many techniques for studying diverse communication networks. Evaluation based on any methodology *other than* actual measurements on actual networks raises questions of fidelity, owing to necessarily simplifications in representing behavior. An effective way to accurately model the behavior of software is to actually run the software [1][5][9][10], by virtualizing the computing platform, partitioning physical resources into different Virtual Environments (VEs), on which we can run unmodified application code [19][22]. However, such emulations typically virtualize execution but not time. The software managing VEs takes its notion of time from the host system's clock, which means that time-stamped actions taken by virtual environments whose execution is multi-tasked on a host reflect the host's serialization. This is deleterious from the point of view of presenting traffic to a network *simulator* which operates in virtual time. Ideally each VE would have its own virtual clock, so that time-stamped accesses to the network would appear to be concurrent rather than serialized.

In this paper, we present a virtual time system that gives virtualized applications running under OpenVZ [24] the temporal appearance of running concurrently on different physical machines. This idea is not completely unique, related approaches have been developed for the Xen [11][12] system. Xen and OpenVZ are very different, and so are the approaches for virtualizing time. Xen is a heavy-weight system whose VEs contain both operating system and application. Correspondingly Xen can simultaneously manage VEs running different operating systems. By contrast, all VEs under OpenVZ (called "containers" in OpenVZ parlance) use and share the host operating system. In Xen virtualization starts at the operating system whereas in OpenVZ virtualization starts at the application. There are tradeoffs of course, we are interested in OpenVZ because it scales better than Xen, as OpenVZ emulation can easily manage many more VEs than can Xen. We believe we are first to introduce virtual time to OpenVZ; by doing so we are able to construct large scale models that run real application code, with rather more temporal accuracy than would be enjoyed without our modifications.

We implement our virtual time system by slightly modifying the OpenVZ and Linux kernels. The OpenVZ modifications measure the time spent in bursts of execution, stop a container on any action that touches the network, and gives one container (the network emulator) control over the scheduling of all the other containers to ensure proper ordering of events in virtual time. Modifications to the Linux kernel are needed to trap interactions by containers with system calls related to time, e.g., if a container calls `gettimeofday()`, the system should return the container's virtual time rather than the kernel's wall clock time – but calls by processes other than OpenVZ's ought to see the kernel's unmodified clock time.

Our time virtualization is not exact. However, comparison with experiments that use real time-stamped data measured on a wireless network reveal temporal errors on the order of 1ms – which is not large for this application. We also measure the overhead of our system's instrumentation and find it to be as low as 3%. In addition, our method is more efficient than the time virtualization proposed for Xen [11]. That technique simply scales real time by a constant factor, and gives each VM a constant sized slice of virtualized time, regardless of whether any application activity is happening. Necessarily, Xen VEs virtualized in time this way can only advance more slowly in virtual time than the real-time clock advances. Our approach is less tied to real time, and in principle can actually

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

advance in virtual time faster than the real-time clock, depending on the number of containers and their applications.

The rest of this paper is organized as follow. Section II reviews related work. Sections III explain our system architecture at a high level, while Section IV provides detailed implementations. Section V evaluates our systems and gives our experimental results. Section VI concludes the whole paper and identifies future work.

II. Related Work

Related work falls into the following three categories: 1) network simulation and emulation, 2) virtualization technique and 3) virtual time systems. They are discussed one by one as follows.

A. Network simulation and emulation

Network simulation and network emulation are two common techniques to validate new or existing networking designs. Simulation tools, such as ns-2 [3], ns-3 [4], J-Sim [5], and OPNET [6] typically run on one or more computers, and abstract the system and protocols into simulation models in order to predict user-concerned performance metrics. As network simulation does not involve real devices and live networks, it generally cannot capture device or hardware related characteristics.

In contrast, network emulations such as PlanetLab [8], ModelNet [9], and Emulab [10] either involve dedicated testbed or connection to real networks. Emulation promises a more realistic alternative to simulation, but is limited by hardware capacity, as these emulations need to run in real time, because the network runs in real time. Some systems combine or support both simulation and emulation, such as CORE [1], ns-2 [3], J-Sim [5], and ns-3 [4]. Our system is most similar to CORE (which also uses OpenVZ), as both of them run unmodified code and emulate the network protocol stack through virtualization, and simulate the links that connect them together. However, CORE has no notion of virtual time.

B. Virtualization technique

Virtualization divides the resources of a computer into multiple separated Virtual Environments (VEs). Virtualization has become increasingly popular as computing hardware is now capable enough of driving multiple VEs concurrently, while providing acceptable performance to each. There are different levels of virtualization: 1) virtual machines such as VMware [20] and QEMU [21], 2) paravirtualization such as Xen [22] and UML [23], and 3) Operating System (OS) level virtualization such as OpenVZ [24] and Virtuozzo [25]. Virtual machine offers the greatest flexibility, but with the highest level of overhead, as it virtualizes hardware, e.g., disks. Paravirtualization is faster as it does not virtualize hardware, but every VE has its own full blown operating system. OS level virtualization is the lightest weight technique among these [18], utilizing the same operating system kernel (and kernel state) for every VE. The problem domain we are building this system to support involves numerous lightweight applications, and so our focus is on the most scalable of these approaches. The potential for lightweight virtualization was demonstrated by Sandia National Lab who demonstrated a one million VM run on the Thunderbird Cluster, with 250 VMs each physical server [2]. While the virtualization techniques used are similar to those of the OpenVZ system we have modified, the Sandia system has neither a network simulator between communicating VMs, nor a virtual time mechanism such as we propose.

C. Virtual time system

Recent efforts have been made to improve temporal accuracy using Xen paravirtualization. DieCast [11], VAN [12] and SVEET [13] modify the Xen hypervisor to translate real time into a slowed down virtual time, running at a slower but constant rate, and they call such mechanism time dilation. At a sufficiently coarse time-scale this makes it appear as though VEs are running concurrently. Other Xen-based implementations like Time Jails [14] enable dynamic hardware allocation in order to achieve higher utilization. Our approach also tries to maximize hardware utilization and keep emulation runtime short. Unlike the mechanism of time dilation, we try to advance virtual clock as fast as possible, regardless it is faster or slower than real time.

Our approach also bears similarity to that of the LAPSE [17] system. LAPSE simulated the behavior of a message-passing code running on a large number of parallel processors, by using fewer physical processors to run the application nodes and simulate the network. In LAPSE, application code is directly executed on the processors, measuring execution time by means of instrumented assembly code that counted the number of instructions executed; application calls to message-passing routines are trapped and simulated by the simulator process. The simulator process provides virtual time to the processors such that the application perceives time as if it were running on a larger number of processors. Key differences between our system and LAPSE are that we are able to measure execution time directly, and provide a framework for simulating any communication network of interest (LAPSE simulates only the switching network of the Intel Paragon).

III. System Architecture

We begin by providing an introduction of OpenVZ system, and then explain the architecture of our system.

A. Overview of OpenVZ

OpenVZ provides container-based virtualization for Linux [24]. It enables multiple isolated execution environments (called *Virtual Environments,* VEs, or *containers*) within a single OS kernel. It provides better performance and scalability as compared with other virtualization technologies. Figure 1 shows the architecture of OpenVZ. A virtual environment looks like a separate physical machine. It has its own process tree starting from the `init` process, its own file system, users and groups, network interfaces with IP addresses, etc. Multiple VEs coexist within a single physical machine, and they not only share the physical resources but also share the same OS kernel. All VEs have to use the same version of the same kernel.

A VE is different from a real OS. A VE uses fewer re-

sources. For example, a newly created Ubuntu VE can have fewer than 10 processes. A VE has limited function compare with a real machine, e.g., it is prohibited from loading or unloading kernel modules inside a VE. Finally, the Linux host operating system provides all kernel services to every VE; the operating system is shared.

OpenVZ offers two types of virtual network interfaces to the VEs, one called a virtual network device (or `venet` in OpenVZ parlance) and the other called a virtual Ethernet device (or `veth` in OpenVZ parlance) [24]. A virtual network device has lower overhead, but with limited functionality, serving simply as a point-to-point connection between a VE and the host OS. It does not have a MAC address, has no ARP protocol support, no bridge support, and no possibility to assign an IP address inside the VE. By contrast, a virtual Ethernet device has slightly higher (but still very low) overhead, but it behaves like an Ethernet device. A virtual Ethernet device consists of a pair of network devices in the Linux system, one inside the VE and one in the host OS. Such two devices are connected via Ethernet tunnel: a packet goes into one device will come out from the other side.

The OpenVZ CPU scheduler has two levels. The first level scheduler determines which VE to give time slice to, according the VE's CPU priority and limit settings. The second level scheduler is a standard Linux scheduler, which decides which process within a VE to run, according to the process priorities.

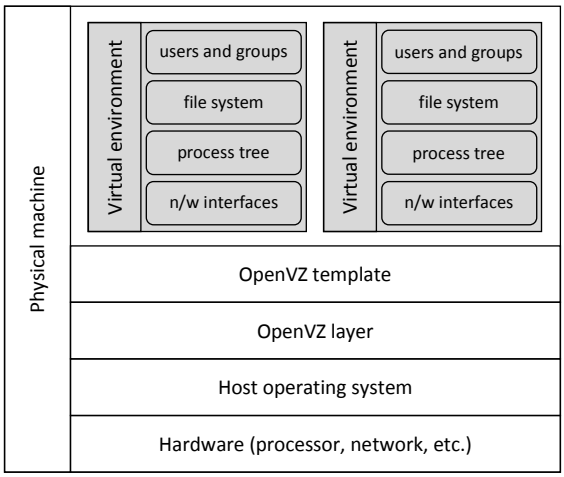


Figure 1 Architecture of OpenVZ

B. System designs

The architecture of our OpenVZ-based emulation system is illustrated by Figure 2. For a given experiment a number of VEs are created, each of which represents a physical machine in the scenario being emulated. Applications that run natively on Linux run in VEs without any modification. The sequencing of applications run on different VEs is controlled by the Simulation/Control application, which runs on host OS (or `VE0` in OpenVZ parlance). Sim/Control communicates to the OpenVZ layer to control VE execution so as to maintain temporal fidelity. For instance, a VE that is blocked on a socket read ought to be released when data arrives on that socket. Sim/Control knows when the data arrives, and so knows when to signal OpenVZ that the blocked VE may run again.

Under unmodified OpenVZ all VEs share the *wallclock* of the host computer (accessed through the shared host operating system). In our virtual time system, each VE has its own *virtual clock* (denoted as $\texttt{vclk}_i$ in Figure 2), while the host OS still uses the wallclock (denoted as `wclk`). Different virtual clocks advance separately, but all of them are controlled by the network simulator via the virtual time kernel module (`V/T module` in the figure).

Sim/Control captures packets sent by VEs and delivers them to destination VEs at the proper time ("proper time" being a function of what happens as the network is simulated to carry those packets). Sim/Control also controls the virtual times of VEs, advancing their virtual times as a function of their execution, but also blocking a VE from running, in order to prevent causal violation. For example, if a packet should arrive at a VE at virtual time t, but the virtual time of that VE is already t+1, a causal violation occurs because the application has missed the packet and may behave differently than expected. Sim/Control is responsible for stopping this VE at virtual time t, until the packet arrives. This is accomplished by modifying the scheduler, as we will describe in Section IV.

Sim/Control consists of two cooperating subsystems: 1) network subsystem (denoted as `N/W` in Figure 2) and 2) virtual time subsystem (denoted as `V/T`). For instance, when a packet sent by VE1 to VE2, it is captured by Sim/Control, which has to know the virtual sending timestamp of that packet in order to know when it entered the network. After the simulator determines the virtual arrival time of the packet at VE2, the simulator must ensure that VE2 has advanced far enough in simulation time to receive that packet, or that VE2 is blocked waiting for a packet, and so needs to be released to run.

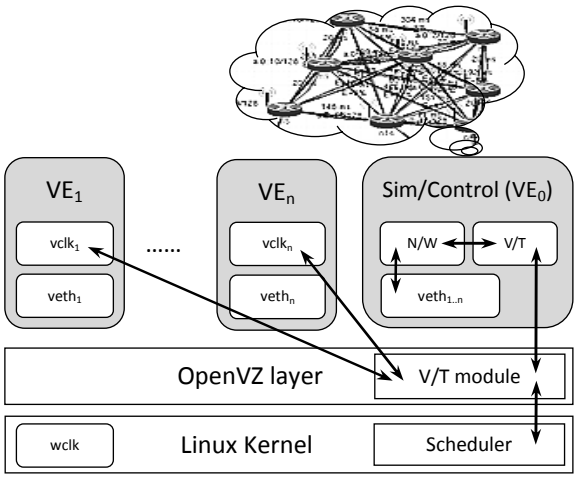


Figure 2 Architecture of network emulation with virtual time

C. Sim/Control

The Sim/Control process captures VE packets, simulates their travel within the imagined network, and delivers them to their destinations. Packet capture is accomplished using the OpenVZ virtual Ethernet device (`veth`). When an application within a VE sends a packet via its `veth` interface, the packet

appears at `veth` in the host OS due to the virtual Ethernet tunneling. Sim/Control monitors all `veth` interfaces to capture all packets sent by all VEs. Similarly, when it wants to deliver a packet to a VE, it just simply sends the packet to the corresponding `veth` interface. The packet tunnels to the VE's corresponding `veth` interface, where the application receives it. The packet travel route is shown in Figure 3.

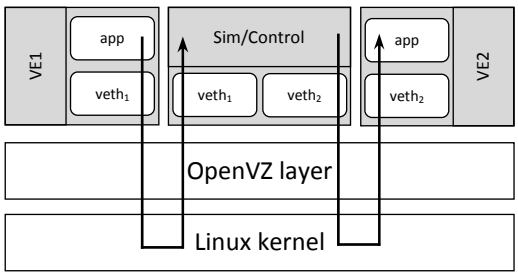


Figure 3 Packet traverse route in emulation

Sim/Control needs to cooperate with the virtual time subsystem when VEs are either sending packets or receiving packets. For example, in real system, blocking socket sends (e.g., `sendto()`) are returned after the packets have been taken care of the underlying OS. Correspondingly, in emulation, the process should perceive comparable amount of elapsed time after the call returns. This is done by trapping those system calls, suspending the VE, have Sim/Control figure out the time at which the packet is taken care, and then return control to the VE, at the corresponding virtual time. Similarly, when an application is blocked waiting to receive a packet, it should be unblocked at the virtual time of the packet's arrival. The comparison between real network operations and emulated ones is shown in Figure 4. As long as the processes perceive comparable elapsed time after system calls are returned and the network simulation gives high enough fidelity for the system measures of interest, this approach is viable. Our approach to integrating the network simulation allows us to include any one of a number of physical layer models. Detailed technical issues are discussed in Section IV.

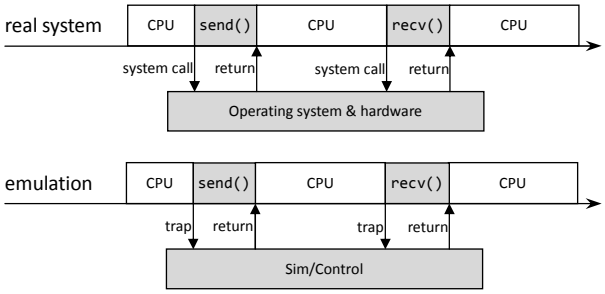


Figure 4 Real network operations vs. simulated network operations

D. Virtual time subsystem

The responsibility of the virtual time subsystem includes advancing virtual clocks of VEs and controlling the execution of VEs. From the operating system's point of view, a process can either have CPU resources and be running, or be blocked and waiting for I/O [30] (ignoring a "ready" state, which rarely exists when there are few processes and ample resources). The wall clock continues to advance regardless of the process state. Correspondingly, in our system, the virtual time of a VE advances in two ways. At the point a VE becomes suspended, the elapsed wallclock time during its execution burst is added to the virtual clock. This is shown in Figure 5.

The situation is different when the application within a VE interacts with the I/O system. Our modified OpenVZ kernel traps the I/O calls, Sim/Control determines the I/O delay and adds that to the VE's virtual clock, and then returns the I/O request to unblock the process. As shown in Figure 5, when I/O delay is accurately simulated, the virtual clock will have the same value as wall clock, and therefore the application perceives the same elapsed time. However, the real elapsed time depends on the time spent on emulating such I/O, which depends on the model and the communication load.

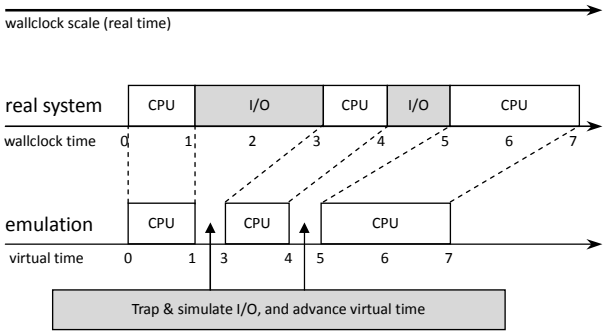


Figure 5 Wall clock time advancement vs. virtual time advancement

It is sometimes necessary to block a running VE in order to prevent casual violation. An example is when an application queries for incoming I/Os, e.g. a non-blocking socket call using `select()` [32]. Even though there may be no pending packets at that *wallclock moment*, it is possible still for a packet to be delivered with a virtual arrival time that is no greater than the virtual time t of the `select()` call, because the virtual clock the sending VE may be less than t. Therefore, when an application makes a non-blocking socket receive call at virtual time t, our system suspends it until we can ensure no packets can arrive with time-stamps less than or equal to t. On a blocking call we need to ensure that the right packet unblocks the VE, and so the same logic applies – before the VE is released at time t, we ensure that any packet with time stamp t or small has already been delivered. Implementation details are given in Section IV.

IV. Implementation

We next present implementation details of our virtual time system, and discuss some related issues.

A. Implementation architecture

As shown in Figure 2, virtual time management needs kernel support, therefore modification to OpenVZ kernel is necessary. We try to keep the modifications simple. The kernel implements only some primitive operations, while Sim/Control calls these operations to control sequencing of

VE execution. Sim/Control runs at user level on the host OS (`VE0`), and communicates with the kernel through new system calls we have implemented. We have chosen system calls to be the communication channel between user space and kernel because of its high efficiency. We placed Sim/Control in user space in order to keep the kernel safe, but the frequent communication between user and kernel raises the question of overheads. Section V.E discusses our measurement of this, found in our experiments to be small.

We first provide the kernel modifications in Subsection B, and then present the design of Sim/Control in Subsection III.C. Finally, we discuss the upper bound of error in Subsection D.

B. Modifications to OpenVZ kernel

OpenVZ scheduler. Scheduling VEs properly ensures the correctness of the emulation. To support this we modified the OpenVZ scheduler so that Sim/Control governs execution of the VEs. By making system calls to the kernel, Sim/Control explicitly gives time slices to certain VEs; a VE may run only through this mechanism.

The typical scheduling sequence of emulation is shown in Figure 6, showing how Sim/Control and VEs take turns running. We refer Sim/Control time slice together with all the subsequent VE time slices before the next Sim/Control time slice as one *emulation cycle*. At the beginning of a cycle, all VEs are blocked. In its time slice Sim/Control pushes all due events to VEs (such as packet deliveries, details in Subsection III.C), and then decides which VEs can have time slices and notifies the kernel that they may run. Causal constraints may leave any given VE blocked, but Sim/Control will always advance far enough in virtual time so that at least one VE is scheduled to run in the emulation cycle. VE executions within an emulation cycle are logically independent of each other, and so their execution order does not matter.

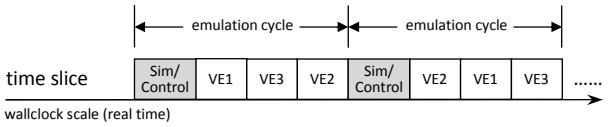


Figure 6 Scheduling of VEs and Sim/Control (example of 3 VEs)

An executing VE is suspended when either it uses up the time slice or when it makes a system call that is trapped (typically one that interacts with the communication system). Such system calls includes network send and receive calls, as discussed in Section III.C. Once a VE makes such special calls, it is blocked immediately and cannot run any more within this emulation cycle. After a VE stops, the actual time it executed will be added to its virtual clock.

This discussion summarizes *Rule #1: A VE can run only after Sim/Control has released it, and will then suspend when either its time slice is consumed, or it executes a trapped system call that interacts with the I/O subsystem.*

Trap network related system calls. We first discuss socket send calls, both blocking and non-blocking. As pointed out by Section III.C, blocking socket sends should be returned after a virtual time equivalent to the time required to transmit the packet. In unmodified OpenVZ, such system calls are returned almost immediately, as the virtual Ethernet device handles packets at an extremely high speed. However, the time elapsed in real systems depends on the underlying physical layer. It can be in the order of microseconds for some gigabit Ethernet [39], but can be as large as several milliseconds for some wireless networks. We modified OpenVZ to give Sim/Control the responsibility of returning those system calls. The VE is suspended at the point of the call, and its current virtual time is updated. Since the packet was presented to the virtual Ethernet interface it tunnels almost immediately to be reflected at its corresponding interface in Sim/Control. Therefore at the beginning of the next emulation cycle Sim/Control observes the packet and marks its send time as the virtual time of its suspended source. Once the packet departure has been fully simulated (and this may take some number of emulation cycles, depending on the network model and the traffic load on the simulator), Sim/Control will know to release the suspended sender. We likewise suspend a VE that makes a non-blocking send, but just to obtain the packet's send time. In this case Sim/Control immediately releases the sender to run again in the following emulation cycle.

Now consider socket receive calls. As discussed in Section III.D, an application that makes a socket receive call (either blocking or not) is suspended *before* it looks at the receive buffer in order to ensure that the state of the receive buffer is correct for that virtual time. The VE must wait at least until the next cycle, at which point Sim/Control will either release it to run, or not, depending on whether there is a threat of violating causality constraints. Once the VE is released to run again it looks at the receive buffer and responds to the previous socket receive call with normal semantics (possibly blocking again immediately, if it is a blocking receive that finds no packet in the buffer).

This discussion summarizes *Rule #2: a VE is always suspended upon making a network related system call.*

Other kernel modifications. Other kernel modifications are also necessary. This includes trapping `gettimeofday()` system calls and returning virtual times to the application, and basing kernel timers on virtual time. The implementations are entirely straightforward and need no further comment.

C. VE Scheduling Control

Sim/Control runs in user-space in the host OS. With the support of the kernel, it only needs to maintain a simple logic to control VE execution. The algorithm used is described in Algorithm 1, and it is a simple variation of a conservative parallel discrete event simulation (PDES) synchronization method [27][28]. Sim/Control maintains its own virtual clock `current_time`. Conceptually, in every time slice of an emulation cycle, Sim/Control does the following one by one: (1) buffers packets sent by VEs during the last cycle (line 10-15), (2) simulates the network and pushes due events to VEs (line 17-23), (3) decides which VEs can run in the next cycle (line 25-31), (4) advances virtual clocks if no VEs can run (line 33-38), and finally (5) yields the processor to let VEs run (line 41-42). In step (3), an event is considered "due" if its virtual

time is no greater than the virtual time of the VE to which it is pushed. We will comment more on this in our section on error analysis.

As shown in Algorithm 1, the Sim/Control calls the network simulator program `nw_sim` to simulate the network (line 17 & 35); this function call needs some explanation. The first parameter is the current status of the network (stored in the `VE` data structure), a status that changes as the simulation executes. The second and third parameters indicate the desired start time and end time of simulation. The fourth parameter is a flag, explained later. The outcomes of `nw_sim` are the events to be returned to VEs (both finished transmissions and packet receptions) within the desired time interval, and they are stored back into the network status `VE`. With this information, the Sim/Control knows which system calls to return and which packets to deliver within a single time slice (line 20-21).

Algorithm 1 Logic for Controlling VE Execution

```
01  for each VE i do
02    init_ve_status(VE[i])
03  end for
04  current_time ← 0
05
06  while true do
07    wait_until_all_ves_stop()
08    last_time ← current_time
09    current_time ← min_i(VE[i].time)
10    for each VE i do
11      for each packet p sent by VE[i] do
12        p.send_time ← VE[i].time
13        buffer_packet(VE[i], p)
14      end for
15    end for
16
17    nw_sim(VE, last_time, current_time, false)
18    while true do
19      for each VE i do
20        return_due_send_calls(VE[i])
21        deliver_due_packets(VE[i])
22        fire_due_timers(VE[i])
23      end for
24
25      for each VE i do
26        VE[i].can_run ← VE[i].ready and
27            VE[i].time < current_time + δ
28      end for
29      if at least one VE can run then
30        break
31      end if
32
33      next_time ← min_j(virtual_timers[j].exp)
34      current_time ←
35          nw_sim(VE, current_time, next_time, true)
36      for each VE i do
37        VE[i].time ← max(VE[i].time, current_time)
38      end for
39    end while
40
41    release_VEs()
42    sched_yield()
43  end while
```

The fourth parameter of `nw_sim` interface is a flag which tells the simulator whether to stop when such return event *first occurs*. When the flag of `nw_sim` is set to false, network is simulated for the given time interval (e.g. line 17) and returns the virtual end-time of its execution period. When the flag is set to true, the network simulator finds the exact time of the next return event (e.g. line 35), stops and returns that time. When the emulator detects that no VEs can run, it advances its current virtual clock to the point when next event happens. Such event can be either a packet transmission or a kernel timer expiration, whichever happens first (line 33-35).

Finally, there is a tunable parameter δ in line 27 used to control how tightly virtual clocks of different VE are synchronized. In the following section we show how an application running multiple threads in a VE can have different behavior in our system that it does in a real system. The smaller δ is, the smaller the potential error of that difference can be. Our experiments use a value of 1msec, which is the minimum possible value in our current platform.

D. Error Analysis

According to Algorithm 1, an event is delivered to a VE *no earlier* than it should be, in the sense that if an event time is t, the VE notices it at a time u that is at least as large as t. Since we suspend a VE at all points where it interacts with the network, if the VE is single-threaded it will be insensitive to the difference between t and u because its behavior is not affected by the arrival at t. This is not the case however if an application is multi-threaded, as we show below.

Consider an application which consists of two threads. Thread A does CPU intensive computation, while Thread B repeatedly receives packets using blocking socket call `recvfrom()`, and calls `gettimeofday()` immediately after it receives a packet. In a Linux system, when there is an incoming packet, Thread B will be immediately be run, pre-empting Thread A. As a result, this application can get the exact time of packet arrival. This is illustrated in Figure 7.

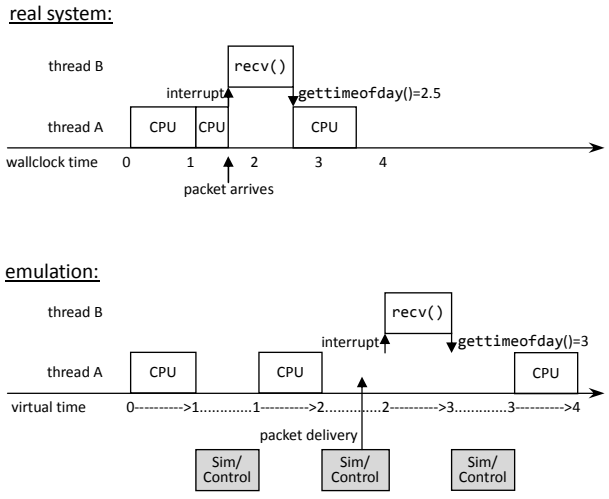


Figure 7 Error in virtual time of packet delivery

Now consider the same application running on our virtual time system. Whenever Thread B makes the socket receive call, the *whole application* – both threads – is blocked by Rule #2. After Sim/Control releases the application to run again, Thread A will take off. However, once the application gets the CPU, it uses the entire time slice because Thread A keeps doing computation. Meanwhile Sim/Control is waiting its turn and so packets are *not* delivered to wake up Thread B until that turn comes around, after the actual delivery time. In this case, the error in that delivery time can be *as most as large* the time slice the application is given to. The comparison is shown in Figure 7.

We summarize the above case as an instance of an *interrupt-like* event. The key problem are situations where in the real system a process is interrupted immediately, whereas in the emulation the interrupting event is not recognized until Sim/Control gets a time-slice. We can reduce this error by reducing the length of the time-slice, but this of course increases the overhead by increasing context switching [30]. The tradeoff between behavioral accuracy and execution speed is a common tradeoff in simulation [29]. We leave exploring such tradeoff as future work.

V. EVALUATION

This section provides our experimental results that demonstrate the performance of our virtual time system.

A. Experimental framework

In order to validate the emulated results, we compare them with that we obtained in [36] within the Illinois Wireless Wind Tunnel (iWWT). The iWWT [35] is built to minimize the impact of environment in wireless experiments. It is an electromagnetic anechoic chamber whose shielding prevents external radio sources from entering the chamber; and whose inner wall is lined with electromagnetically absorbing materials, which reflect minimal energy. This gives us a simulated "free space" inside the chamber, which is ideal for conducting wireless network experiments.

We run the same application as we used in [36] within our emulator. We notice the hardware difference between the machine on which we run our emulator, and the devices we used to collected data inside the chamber. Specifically, our emulator is running on a Lenovo T60 laptop with Intel T7200 CPU and 2GB RAM (if not otherwise specified), while we used Soekris Engineering net4521 [37] with mini-PCI wireless adapters plugged in as wireless nodes inside the iWWT. Due to the difference in processors, applications running on the Soekris box should observe longer elapsed times than the Lenovo laptop. However, this should not bring large error into the results, as the applications we run are I/O bound ones, i.e. the time spend on I/O is dominant while the time spend on CPU is negligible.

In the experiment inside the chamber, the wireless nodes were operating on 802.11a [38]. Correspondingly, we have an 802.11a physical layer model in our Sim/Control, which predicts packet losses and delay. Our 802.11a model implements CSMA/CA, and it uses bit-error-rate model to stochastically sample lost packets.

B. Validating bandwidth, delay and jitter – one-link scenario

We start with the simplest scenario with two wireless nodes and one wireless link. Packets are sent to the receiver as fast as possible using 54Mbps data rate under 802.11a, and the receiver records the timestamp of each received packet. The comparison between real trace and emulated results is shown in Figure 8 and Figure 9. We study the delay rather than throughput because the sender is sending packets of constant size, and therefore accurate delay implies accurate throughput, but not vice versa. The experiment actually persists for 10sec in real time, but we only show 20 packets for conciseness.

As shown in Figure 8, the emulated result (the 1-flow line) under our virtual time system is almost identical to the real trace, with the error within 1msec. The only difference is due to random retransmissions, which are caused by low SINR. In 802.11a, when the receiver cannot correctly decode the data frame, it will not send an ACK frame. In this case, the sender will have to retransmit that data frame, and this will approximately double the transmission time of this single packet compare with no retransmission. From the figure, we are able to tell when a retransmission happens by examining inter-packet arrival time. As retransmission is a random event, it is reasonable that our 802.11a model cannot predict retransmission for the exact packet as real trace. In general, the model predicts comparable retransmission rate.

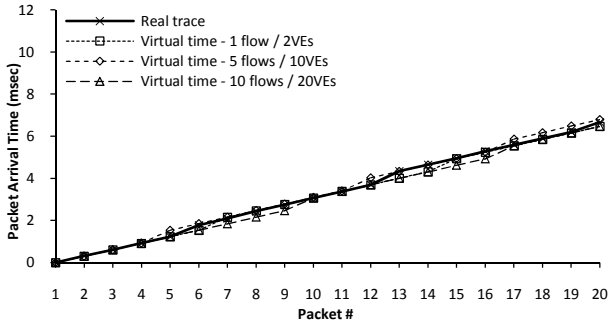


Figure 8 Packet arrival time, one link scenario, with virtual time

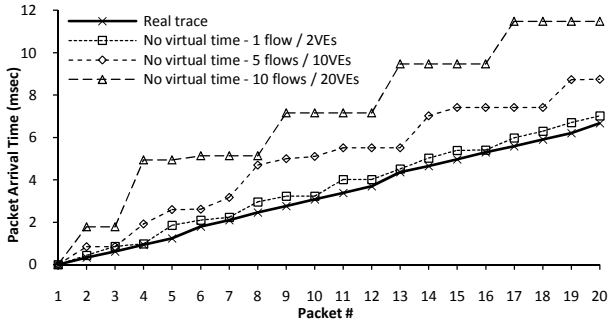


Figure 9 Packet arrival time, one link scenario, without virtual time

To demonstrate the accuracy of our system under different loads, the previous one-link scenario is replicated several times, with each replication emulated simultaneously. However, replicated links are only used to saturate the emulator, so

they are independent and will not interfere with each other. This represents the scenario in which we have multiple non-interfering wireless links; each flow has the same behavior as a single flow (modulo differences in random number seeds.) In Figure 8, the 5-flow line and the 10-flow line show the results of a single flow out of the 5 or 10 simultaneous ones. Regardless of the number of flows, the behavior of each flow is identical to 1-flow case, as expected.

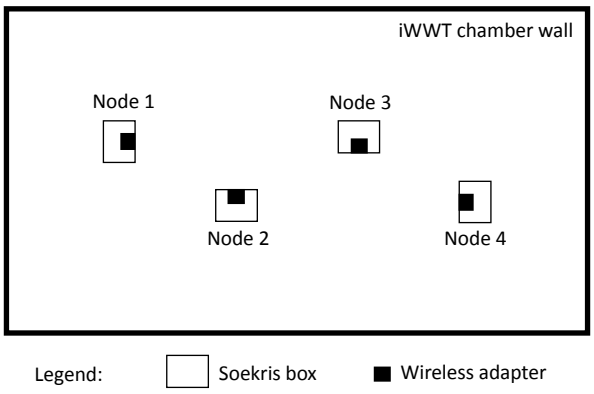
For comparison, we run the same experiment under the system without virtual time, and the results are shown in Figure 9. The 1-flow result is has only slightly larger error than that with virtual time, being caused by scheduling delay. This occurs when a VE cannot get the CPU and execute exactly on time, including (1) when the network emulator should deliver a packet to destination VE, and (2) when the destination VE should process an incoming packet. In fact, such scheduling delay also exists in virtual time implementation, but VEs will not perceive delay because their virtual clocks do not advance meanwhile. The error of not having virtual time here is as small as 1msec, but it will scale up when the number of VEs increases, as more VEs may introduce longer delay. As shown by the 5-flow line in Figure 9, the error can be as large as several milliseconds. Worse still, when the offer load is too high (e.g. the 10-flow line) for the emulator to process in real time, the error accumulates and becomes larger and larger. This is occurs because the emulator cannot run fast enough to catch up with real time.

C. Validating bandwidth, delay and jitter – two-link scenario

We next validate the scenario with four wireless nodes and two conflicting wireless links. Space limitations inside the iWWT prohibit study of more complex scenarios. Figure 10 shows the experimental setup. Although we only have four wireless nodes, we have six different combinations by changing source/destination pair and data rates. For example, Scenario 3 and Scenario 4 are those of heterogeneous data rate.

We use the `iperf` link test application [34] to test both links, and we are interested in both bandwidth and jitter (delay variation). We configure the emulator to simulate the above scenario. However, by examining the real data, we found that the link with Node 2 as sender usually has a higher throughput, regardless of its receiver. We conclude this is due to the hardware, although the four wireless adapters are of the same manufacture and same model. We capture such hardware characteristic by increasing the antenna gain of Node 2 in our network simulator. Higher antenna gain results in higher SINR, lower bit error rate (BER), lower retransmission rate, and finally higher throughput.

The results are shown in Figure 11 and Figure 12. The former shows comparison of throughput and the latter shows that of jitter. We observe a very accurate emulated throughput, with error less than 3% in most cases. On the other hand, the jitter obtained from emulated platform has a larger error, which is within 10% in most cases but which sometimes is as large as 20%. By comparing timestamps generated by the simulator and that perceived by the application, we found such error is due to the inaccuracy of the 802.11a model, *not* the virtual time system itself. As discussed before, accurate delay implies accurate throughput but not vice versa. The results here demonstrate that jitter is more difficult to model than throughput.



Scenario ID	Link 1		Link 2	
	src→dst	data rate	src→dst	data rate
1	2→1	6Mbps	3→4	6Mbps
2	2→4	6Mbps	3→1	6Mbps
3	2→1	54Mbps	3→4	6Mbps
4	2→4	54Mbps	3→1	6Mbps
5	2→1	54Mbps	3→4	54Mbps
6	2→4	54Mbps	3→1	54Mbps

Figure 10 Two-link experiment setup inside the iWWT

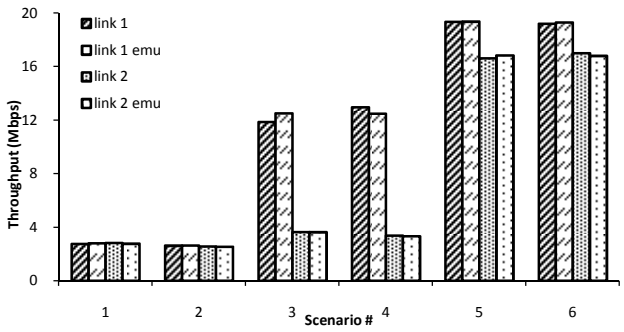


Figure 11 Two-link scenarios – throughput

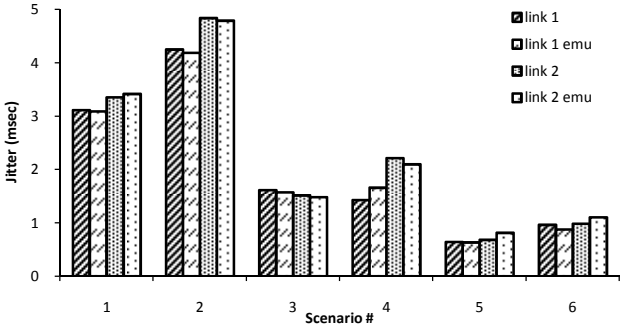


Figure 12 Two-link scenarios – jitter

D. Emulation runtime and scalability

We tested the execution speed of our system, by modifying the number of simultaneous network flows. We reuse the previous one-link scenario, in which the sender transmits packets to the receiver for 10sec using 54Mbps data rate. As in Subsection B, we replicate this link by several times, but only in order to saturate the emulator.

The emulation runtime under different loads is shown in Figure 13. When there is only one link, the runtime is only less than 2sec in real time. Compared with 10sec elapsed in virtual time, emulation runs faster because in this case the emulated I/O is faster than real I/O. As the number of flows increases, the runtime increases linearly as well. When there are more than 7 flows, the emulation runs slower than real time because of large volume of traffic. In addition, we plot the CPU usage percentage, and we find that our system can achieve high CPU usage when there is enough load. It cannot achieve 100% CPU usage because our Lenovo laptop is using dual-core CPU. As explained in Section IV.C, before the emulator process starts its time slice, it will wait until all VEs stop running. Different VEs can run on different CPUs simultaneously, but the emulator process runs on only one CPU, and meanwhile the other CPU is idle. As shown in Figure 13, increased number of flows results in higher CPU utilization. This is because as the number of flows increases and so does the number of VEs, the fraction of time during which the CPUs are saturated by VE execution also increases.

To demonstrate the scalability of our system, we tested our system on a Dell PowerEdge 2900 server with dual Intel Xeon E5405 and 16GB RAM. We rerun the above scenario but with 160 flows and 320 VEs, which finishes in 307sec (compared with < 2 sec. for 1 flow). Our current system can only run on a single machine, but with its distributed version as a future work, we will be able to emulate more network nodes. Nevertheless, we found that implementation with OpenVZ yields high VE density (320 VEs per physical machine), thanks to the light weight of OpenVZ.

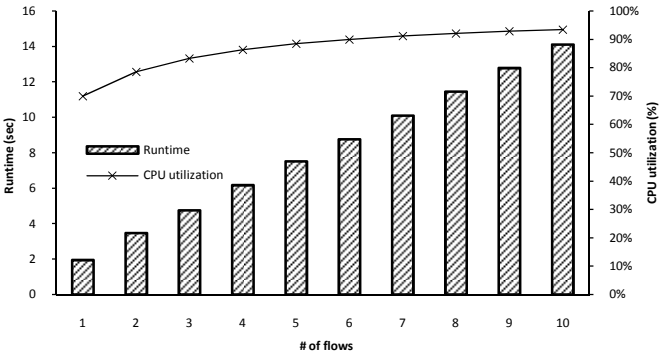


Figure 13 Emulation runtime

E. Implementation overhead

We are concerned about the implementation overhead of our system, and we measure it in the following way. We slightly modify the user-level emulator application, making it run in real time instead of virtual time so that it can run on original Linux platform. When running on original Linux, we use the `snull` virtual Ethernet tunnel module [33] to replace the virtual Ethernet devices created OpenVZ, so that both `iperf` and the emulator can be configured in the same way.

We reuse the scenario of the previous subsection, but we only use 5 flows so that our Lenovo laptop is capable to run in real time. When running in real time, either on original Linux or on original OpenVZ, the runtime of emulation is exactly 10sec. Instead of using the elapsed time of wallclock, we use the total CPU time for comparison. For instant, if the average CPU utilization is 60% during the 10-second period, the total CPU time is 6sec.

	Original Linux	Original OpenVZ	Virtual time system
Total CPU time	6.345 sec	6.478 sec	6.654 sec
Time in %	100.0%	102.1%	104.9%

Figure 14 Implementation overhead

The comparison among Linux, OpenVZ, and our virtual time system is shown in Figure 14. For the scenario we consider, we obverse 2% overhead of OpenVZ compared with original Linux. This matches the claim on OpenVZ project website, which says OS-level virtualization only has 1%-3% overhead [24]. In addition, we observe another 3% overhead of implementing virtual time, based on the original OpenVZ system. We find that such overhead is due to both frequent system calls and context switches. Implementing the emulator in kernel space might help on reducing such overhead, but we prefer to keep the kernel safe and simple. The observed 3% overhead is low, and considering the performance of OS-level virtualization, we conclude that our system is highly scalable.

Finally, we notice that such implementation overhead on OpenVZ is competitively low compared with other virtualization techniques. For instance, QEMU without a kernel acceleration module is 2X to 4X slower [21]. We are currently developing a virtual time system for QEMU as well, in order to support time virtualization to applications running on a larger number of operating systems.

VI. Conclusions and Future Work

We have implemented a virtual time system which allows unmodified application to run on different virtual environments (VEs). Although multiple VEs coexist on a single physical machine, they perceive virtual time as if they were running independently and concurrently. Unlike previous work based on Xen paravirtualization, our implementation is based on OpenVZ OS-level virtualization, which offers better performance and scalability (at the price of less flexibility). In addition, our system can achieve high utilization of physical resources, making emulation runtime as short as possible. Our implementation has only 3% overhead compared with OpenVZ, and 5% compared with native Linux. This indicates that our system is efficient and scalable.

Through evaluation, we found the accuracy of virtual

time can be within 1msec, at least if an accurate network simulator is used. It is indeed the model that introduces the error, rather than the virtual time system itself. While our system can currently simulate limited type of hardware, future work can model more hardware configuration, e.g. multicore CPU, other I/O devices such as disk. It is also worth exploring the tradeoff between behavioral accuracy and execution speed in the domain of emulation using unmodified compiled code.

ACKNOWLEDGEMENTS

This material is based upon worked supported under Dept. of Energy under Award Number DE-0E00000097, and under support from the Boeing Corporation.[1]

REFERENCES

[1] J. Ahrenholz, C. Danilov, T.R. Henderson, and J.H. Kim, "Core: a real-time network emulator", MILCOM 2008, Nov. 2008.

[2] J. Mayo, R. Minnich, D. Rudish, R. Armstrong, "Approaches for scalable modeling and emulation of cyber systems: LDRD final report", Sandia report, SAND2009-6068, Sandia National Lab, 2009.

[3] The Network Simulator - ns-2: http://www.isi.edu/nsnam/ns/

[4] The ns-3 project. http://www.nsnam.org/

[5] A. Sobeih, W.-P. Chen, J. Hou, L.-C. Kung, N. Li, H. Lim, H.-Y. Tyan, and H. Zhang, "J-Sim: a simulation and emulation environment for wireless sensor networks", IEEE Wireless Communications Magazine, vol. 13, no. 4, pp. 104-119, Aug. 2006.

[6] OPNET Modeler: Scalable Network Simulation: http://www.opnet.com/solutions/network_rd/modeler.html

[7] P. Riley and G. Riley, "SPADES--A distributed agent simulation environment with software-in-the-loop execution", Winter Simulation Conference Proceedings, pp. 817–825, 2003.

[8] B. Chun., D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services", SIGCOMM Computer Communication Review, 2003.

[9] A. Vahdat, K. Yocum, K.Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker, "Scalability and accuracy in a large-scale network emulator", In Proceedings of the 5th Symposium on Operating Systems Design and Implementation, 2002.

[10] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks", In Proceedings of the 5th Symposium on Operating Systems Design and Implementation, 2002.

[11] D. Gupta, K. Vishwanath, and A. Vahdat, "DieCast: testing distributed systems with an accurate scale model", In Proceedings of the 5th USENIX Symposium on Networked System Design and Implementation (NSDI'08), San Francisco, CA, USA, 2008.

[12] P. Biswas, C. Serban, A. Poylisher, J. Lee, S.-C. Mau, R. Chadha, C.-Y. Chiang, R. Orlando, K. Jakubowski, "An integrated testbed for virtual ad hoc networks", Proceedings of TRIDENTCOM 2009, April 6-8, 2009.

[13] M. Erazo, Y. Li, and J. Liu, "SVEET! A scalable virtualized evaluation environment for TCP", TridentCom'09, 2009.

[14] A. Grau, S. Maier, K. Herrmann, K. Rothermel, "Time Jails: a hybrid approach to scalable network emulation", Workshop on Principles of Advanced and Distributed Simulation (PADS), pp. 7-14, 2008.

[15] E. Weingärtner, F. Schmidt, T. Heer, and K. Wehrle, "Synchronized network emulation: matching prototypes with complex simulations", SIGMETRICS Perform. Eval. Rev., vol. 36, no. 2, pp. 58–63, 2008.

[16] R. Pan, B. Prabhakar, K. Psounis, and D. Wischik, "SHRiNK: a method for scalable performance prediction and efficient network simulation", In IEEE INFOCOM, 2003.

[17] P. Dickens, P. Heidelberger, and D. Nicol, "A distributed memory LAPSE: parallel simulation of message-passing programs", In Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS '94), pp. 32-38, Jul. 1994.

[18] P. Padala, X. Zhu, Z.Wang, S. Singhal, and K. Shin, "Performance evaluation of virtualization technologies for server consolidation", Technical Report HPL-2007-59, HP Labs, Apr. 2007.

[19] B. Walters, "VMware virtual platform", Linux journal, 63, Jul. 1999.

[20] VMware virtualization software: http://www.vmware.com/

[21] F. Bellard. "QEMU, a fast and portable dynamic translator", Proceedings of the USENIX Annual Technical Conference, FREENIX Track, pp. 41–46, 2005.

[22] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization", In Proceedings of the nineteenth ACM symposium on Operating Systems Principles (SOSP19), pages 164–177. ACM Press, 2003.

[23] The User-Mode Linux Kernel: http://user-mode-linux.sourceforge.net/

[24] OpenVZ: a container-based virtualization for Linux. Project website available at http://wiki.openvz.org/Main_Page

[25] Virtuozzo Containers: http://www.parallels.com/products/pvc46/

[26] G. Bhanage, I. Seskar, Y. Zhang, and D. Raychaudhuri, "Evaluation of OpenVZ based wireless testbed virtualization", Technical Report, WINLAB-TR-331, Rutgers University, 2008.

[27] R. Fujimoto, "Parallel discrete event simulation", Communication of the ACM, vol. 33, no. 10, pp. 30-53, 1990.

[28] D. Nicol, "The cost of conservative synchronization in parallel discrete event simulations", Journal of the ACM, vol. 40, pp. 304-333, 1993.

[29] D. Nicol, "Tradeoffs between model abstraction, execution speed, and behavioral accuracy", European Modeling and Simulation Symposium, 2006.

[30] A. Tanenbaum, Modern Operating Systems, Third Edition, Prentice Hall, Dec. 2007.

[31] D. Bovet and M. Cesati, Understanding Linux Kernel, Third Edition, O'Reilly Media, Nov. 2005.

[32] C. Benvenuti, Understanding Linux Network Internals, O'Reilly Media, Dec. 2005.

[33] J. Corbet, A. Rubini, and G. Kroah-Hartman. Linux Device Drivers, Thrid Edition. O'Reilly Media, Feb. 2005.

[34] Iperf link test application: http://iperf.sourceforge.net/

[35] N. Vaidya, J. Bernhard, V. Veeravalli, P. R. Kumar , R. Iyer, "Illinois Wireless Wind Tunnel: a testbed for experimental evaluation of wireless networks", SIGCOMM'05 Workshops, 2005.

[36] Y. Zheng and N. Vaidya, "Repeatability of Illinois Wireless Wind Tunnel", Technical Report, Wireless Networking Group, University of Illinois at Urbana-Champaign, May 2008.

[37] Soekris Engineering box: http://www.soekris.com/net4521.htm

[38] 802.11a-1999 High-speed Physical Layer in the 5 GHz band. IEEE standard, 1999.

[39] D. Jin, D. Nicol, and M. Caesar, "Efficient gigabit ethernet switch models for large-scale simulation", Principles of Advanced and Distributed Simulation (PADS), May 2010.

[40] Y. Zheng and D. Nicol, "Validation of radio channel models using an anechoic chamber", Principles of Advanced and Distributed Simulation (PADS), May 2010.

[1] This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

Efficiently Scheduling Multi-core Guest Virtual Machines on Multi-core Hosts in Network Simulation

Srikanth B. Yoginath and Kalyan S. Perumalla
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
yoginathsb@ornl.gov, perumallaks@ornl.gov

***Abstract*—Virtual machine (VM)-based simulation is a method used by network simulators to incorporate realistic application behaviors by executing actual VMs as high-fidelity surrogates for simulated end-hosts. A critical requirement in such a method is the simulation time-ordered scheduling and execution of the VMs. Prior approaches such as time dilation are less efficient due to the high degree of multiplexing possible when multiple multi-core VMs are simulated on multi-core host systems. We present a new simulation time-ordered scheduler to efficiently schedule multi-core VMs on multi-core real hosts, with a virtual clock realized on each virtual core. The distinguishing features of our approach are: (1) customizable granularity of the VM scheduling time unit on the simulation time axis, (2) ability to take arbitrary leaps in virtual time by VMs to maximize the utilization of host (real) cores when guest virtual cores idle, and (3) empirically determinable optimality in the tradeoff between total execution (real) time and time-ordering accuracy levels. Experiments show that it is possible to get nearly perfect time-ordered execution, with a slight cost in total run time, relative to optimized non-simulation VM schedulers. Interestingly, with our time-ordered scheduler, it is also possible to reduce the time-ordering error from over 50% of non-simulation scheduler to less than 1% realized by our scheduler, with almost the same run time efficiency as that of the highly efficient non-simulation VM schedulers.**

Keywords—Virtual Machines, Virtual Clocks, Scheduling, Hypervisors, Network Simulation, Network Emulation

I. INTRODUCTION

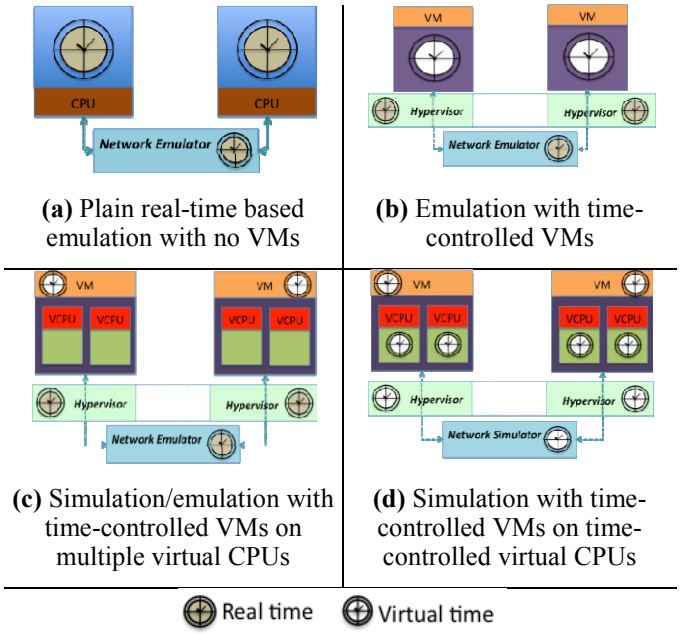
Historically, network simulators realized approximations of the application layer via software models of the same and integrating them with models of the lowers layers such as transport-, network- and physical layers. Since the software models were part of the network simulator timestamp-ordered discrete event simulation of their overall integrated execution was straightforward. Within the last decade, the application layer models have been greatly enhanced in fidelity via the use of virtual machine (VM) technologies to incorporate actual, complete operating system (OS) instances to serve as end-host behaviors. The integration of OS instances via VM platforms into the rest of the network simulator raised the issue of real- and virtual time integration, which was largely solved by the use of approaches such as time dilation. Such approaches have been adequate for single-processor virtual machines multiplexed on single-processor host machines. More recently, with the emergence of multi-core host systems, the issue of time control and time integration has resurfaced with the added dimension of multi-core considerations.

Consider a network simulator that employs multiple host (computing) nodes to host multiple guest (simulated) nodes. The guest nodes are realized via virtual machines that act as surrogates of simulated end-host nodes in the simulated scenario. The issue at hand is that both the host nodes as well as the guest nodes are multi-core systems, whose simulation time advances must be controlled and coordinated to deliver virtual time-ordered integrated execution with the rest of the network simulator. Native VM schedulers are clearly inappropriate because they are designed with processor utilization and fairness considerations that are fundamentally different from virtual time-ordered scheduling. While time dilation [1] approaches can be employed in the multi-core setting as well, they are fundamentally sub-optimal in their host processor utilization when guest virtual cores idle in the scenarios. This sub-optimality can become more pronounced when the number of cores in either the guest and/or the host system is large. Our present work explores VM integration into network simulators by generalizing the virtual time representations taken down to the level of each virtual core.

Figure 1 shows our approach in relation to a few past approaches: (a) represents free-running emulations with only real time-based clocks, (b) represents VMs integrated into emulations via virtual clocks controlled with approaches such as time dilation, (c) shows multi-core execution with one virtual clock per VM, and (d) shows a distinct virtual clock for every entity down to the level of each virtual core (VCPU). The scheme shown in Figure 1(d) is the focus of this paper.

Here, we are interested in supporting the capability to simultaneously schedule multiple single-core and/or multi-core VMs in simulation-time order on native multi-core hardware. Unlike schedulers that maintain only one virtual clock per VM, our scheduler supports a separate virtual clock for each virtual core (VCPU) of a multi-core VM, maintained with its own simulated timeline as a Logical Process (LP) in the simulation framework, making it amenable for incorporation into a discrete-event based simulation system.

978-1-4577-1365-1/11/$26.00 ©2011 IEEE



(a) Plain real-time based emulation with no VMs

(b) Emulation with time-controlled VMs

(c) Simulation/emulation with time-controlled VMs on multiple virtual CPUs

(d) Simulation with time-controlled VMs on time-controlled virtual CPUs

Figure 1: Real-time vs. virtual-time representations in a few simulation/emulation approaches

A. *Organization*

The rest of the article is organized as follows. A brief overview of virtualization, its related terminology, and its use in network simulation are provided in the next section. The implementation details of our scheduler are documented in Section III, followed by a test setup description in Section IV, and a performance analysis in Section V. The findings are summarized in Section VI, related work discussed in Section VII, and future work is presented in Section VIII.

II. BACKGROUND AND TERMINOLOGY

A. *Virtualization*

Virtualization replaces the OS to be the lowest level software running directly over the hardware. By doing so it segregates the hardware and the OS by virtualizing the hardware components to the OS, and relieves the tight hardware-OS coupling. As an important consequence, OS instances from different vendors co-exist and run on the same physical resource, interacting with the virtual counterparts of the physical hardware. The thin multiplexing layer between the hardware and the OS instances is called the *hypervisor*.

Generally there are two types of virtualization: Full-Virtualization (FV), and Para-Virtualization (PV). In FV, the hypervisor supports any unmodified guest OS to run. In PV, the hypervisor provides hypercalls for accessing any hardware service and this requires the modification of a guest OS.

B. *The Xen Hypervisor*

The Xen hypervisor [2] is a popular, powerful open source industry standard for virtualization, supporting a wide range of architectures including x86, x86-64, IA64, and ARM, and guest OS types including Windows®, Linux®, Solaris® and various versions of BSD OS [3]. Although Xen started as a PV, FV is supported using QEMU, a generic machine emulator and virtualizer[4].

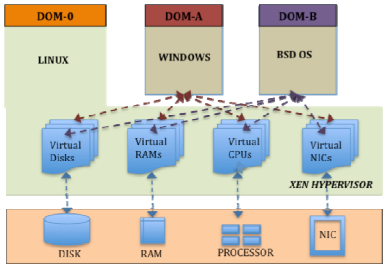


Figure 2: Xen architecture

Figure 2 shows a schematic of guests running on the Xen hypervisor. The guest OS in Xen terminology is referred to as Guest Domain or simply DOM. Each DOM has a unique identifier called its Domain ID (DOM-ID). The first DOM called DOM0 is a privileged one with special management privileges. System administration tasks such as suspension, resumption, and migration of DOMs are managed via DOM0.

In a multi-core system, a software interface called Virtual Central Processing Unit (VCPU) is supported by Xen to provide a virtual equivalent of a processor core. Another interface called Physical CPU (PCPU) maps to an actual hardware core. Thus, all PCPUs are contained within Xen, and the number of PCPUs, Cp, is equal to the number of physical cores (across all processor sockets on the machine). Each DOMi uses a number of VCPUs, Ci, as configured by the user. In network simulations, it is desirable to have $Cp << \sum Ci$, i.e., the number of host cores is much smaller than the total number of virtual cores being multiplexed, in order to be able to host a large number of simulated end-hosts on relatively small simulation test-bed hardware. Also, $Cp \geq Ci$ for every DOM i, i.e., the number of virtual cores in any individual DOM is smaller than the number of physical cores available, because the test-bed hardware is usually a high-end system with many cores than an end-host in a typical simulated network scenario.

C. *NetWarp*

NetWarp, is a scalable computer network simulation framework being developed at our institution to utilize a large number of multi-VCPU VMs sustained on multi-core nodes interoperating with a packet-level backbone network simulation also executing in parallel. The execution architecture thus presents two distinct components of time-controlled execution, namely, the control of intra-node pacing among VMs within a multi-core node, and the control of inter-node pacing for virtual time coordination across multiple multi-core nodes. In this article, we focus on the first component of NetWarp, namely, time-controlled execution of multi-VCPU VMs on multi-core host nodes.

D. *Design Principles*

The problem of scheduling multi-VCPU DOMs on a multi-core system is defined by the following considerations when used in network simulation.

The DOMS are bound together as one simulation application requiring that the simulation time must be global across all the DOMs, and the simulation time must account for the idle-VCPU time within a DOM. A strict time-order in scheduling the VCPUs for execution must be maintained. The scheme must also allow addition and removal of DOMs

dynamically in the simulation without adversely affecting the performance of the running simulation.

In Xen, the scenario network setup, interaction with other DOMs and the monitoring of their execution are performed by DOM0. Also, the device of communication (bridge or router) to which all the other DOMs connect is serviced by DOM0. Hence, DOM0 must be given sufficient cycles to ensure that interactivity is not compromised. Yet, DOM0 must not starve other DOMs.

A *tick* is the time allotted for a particular VCPU when it is scheduled for execution. The chosen tick duration must ensure low run time: the smaller the tick size, the better is the ability to interleave execution among VCPUs. However, this also increases the number of scheduling operations, which can impose larger overhead.

To accommodate these considerations, we create a new scheduler for Xen hypervisor.

III. IMPLEMENTATION

We refer to the NetWarp scheduler for Xen as NSX and the native scheduler of Xen, called the Credit Scheduler[2,5,7], as CSX. In this section, we describe the implementation details of NSX to meet our needs as outlined in the preceding section, and provide a comparison to the internals of CSX.

A. *Scheduler Structure in Xen*

Scheduling in Xen shares some concepts with an operating system that provides an *N:M* threading library. In such a system, the operating system kernel schedules *N* threads (typically one per physical context). Upon each OS thread, a user space library multiplexes into *M* user-space threads. In Xen, the OS threads are analogous to the VCPUs and the user-space threads represent processes within the domain [5]. Essentially, there exist three scheduler tiers in Xen:

(a) User-space threading library maps the user-space threads to kernel threads (with in a DOM)
(b) Guest DOM OS maps its kernel threads to VCPUs
(c) Hypervisor maps each VCPU to a physical CPU (PCPU) dynamically at run time.

Both CSX and NSX are ultimately concerned with mapping VCPUs to PCPUs. A new scheduler such as NSX can replace the CSX relatively easily due to the modular design of Xen. The Xen design prescribes a set of functions that needs to be implemented and interfaced by the new scheduler. Hence, the CSX implementation can be viewed separately from Xen's internals and schedulers with different strategies for accounting can be realized in a similar way.

By reassigning the relevant function pointers to the scheduler's interface variables, our scheduler functionality is integrated into Xen. For example, the *init_domain* variable is a function pointer to *csched_dom_init*, which is used for the initialization of a DOM. After integrating our scheduler, Xen invokes our routines for decisions on scheduling any VCPU onto a PCPU.

B. *NetWarp Scheduler Data structures*

Figure 3 shows the static object created for the NSX to interface with Xen. Similar to CSX, we maintain four different data-structures in NSX, namely,

a. *nw_pcpu* – correspond to the PCPUs
b. *nw_vcpu* – correspond to the VCPUs
c. *nw_dom* – corresponds to a DOM
d. *nw_private* – is global data shared by all DOMs.

```
struct scheduler sched_nw_def = {
      .name             = "SMP Netwarp
Scheduler",
      .opt_name         = "nw",
      .sched_id         = XEN_SCHEDULER_NW,
      .init_vcpu        = nw_vcpu_init,
      .destroy_vcpu     = nw_vcpu_destroy,
      .init_domain      = nw_dom_init,
      .destroy_domain   = nw_dom_destroy,
      .sleep            = nw_vcpu_sleep,
      .wake             = nw_vcpu_wake,
      .adjust           = nw_dom_cntl,
      .pick_cpu         = nw_cpu_pick,
      .do_schedule      = nw_schedule,
      .init             = nw_init,
      .tick_suspend     = nw_tick_suspend,
      .tick_resume      = nw_tick_resume,
};
```

Figure 3: NSX scheduler interface

The *nw_pcpu* data-structure as shown in Figure 4, comprises a VCPU queue called *runq*, a *timer*, and a counter named *tick* that keeps a count of the number of ticks used. The number of instances of the *nw_pcpu* data structure corresponds to the number of processor cores in the physical hardware. The next VCPU to schedule on a particular physical core is chosen by the Xen scheduler from the *runq* list of the *nw_pcpu* object corresponding to that physical core.

```
struct csched_pcpu {
   struct list_head runq;
   uint32_t runq_sort_last;
   struct timer ticker;
   unsigned int tick;
};
```

```
struct nw_pcpu{
   struct list_head runq;
   struct timer ticker;
   unsigned int tick;
};
```

Figure 4: CSX's PCPU vs. NSX's PCPU structures

Figure 5 compares data-structures *csched_vcpu* and *nw_vcpu* (some variables concerned with keeping a log of VCPU status in *csched_vcpu* data-structure are not included in Figure 5 for clarity purposes). As can be seen most of the variables in *nw_vcpu* are the same as in *csched_vcpu*, except that the *credit* and *pri* (priority) variables of the *csched_vcpu* are replaced by the *nticks* variable of *nw_vcpu* and the *ref_time*. The *nticks variable* keeps track of a number of ticks that the VCPU has used up, and the *ref_time* gives a reference-time from which *nticks* are tracked. The *ref_time* along with the *nticks* gives the Local Virtual Time (*LVT*) as $LVT = ref\_time + (nticks \times tick\_size)$ for a VCPU.

```
struct csched_vcpu {
   struct list_head runq_elem;
   struct list_head active_vcpu_elem;

   struct csched_dom *sdom;
   struct vcpu *vcpu;
   atomic_t credit;
   uint16_t flags;
   int16_t pri;
      …
};
```

```
struct nw_vcpu{
   struct list_head runq_elem;
   struct list_head active_vcpu_elem;
   struct nw_dom *sdom;
   struct vcpu *vcpu;
   uint16_t flags;
   atomic_t nticks;
   s_time_t ref_time;
};
```

Figure 5: CSX's VCPU vs. NSX's VCPU structures

Each element of *runq* of an *nw_pcpu* is of type *runq_elem*. As the *runq_elem* is related to the *runq* of the *nw_pcpu* object, in a similar way as the *active_vcpu_elem* object is related to the *nw_dom*'s (discussed next) *active_vcpu* queue. Both these data-structures aid the inserting and removing, to and from their corresponding queues by manipulating the *prev* and *next* pointer values.

```
struct csched_dom {
   struct list_head active_vcpu;
   struct list_head active_sdom_elem;

   struct domain *dom;
   uint16_t active_vcpu_count;
   uint16_t weight;
   uint16_t cap;
};
```

```
struct nw_dom{
   struct list_head active_vcpu;
   struct list_head active_sdom_elem;
   struct domain *dom;
   uint16_t active_vcpu_count;
   spinlock_t lock;
   s_time_t dom_lvt;
};
```

Figure 6: DOM variables in CSX vs. NSX

Figure 6 shows the CSX's data-structure *csched_dom* and NSX's *nw_dom* for domains. The *nw_dom* comprises a list named *active_vcpu*, which is a list of *nw_vcpu* belonging to this DOM. It contains a member-variable named *active_sdom_elem*, which is initialized to point to self, and aids in the functioning and operation of the *active_sdom* queue in the *nw_priv* global object of type *nw_private*. The *nw_priv* holds all the active DOMs in this *active_sdom* queue.

Xen maintains an object of type *domain* for each of the DOMs. The *nw_dom*'s *dom* pointer points to this data-structure object. The *nw_dom* comprises an integer variable that holds a record of active number of VCPUs in the DOM. It also has a member variable *dom_lvt* to keep track of the DOM's simulation time (max LVT value of all the VCPU's of this DOM) and a *spin-lock* used by the DOM's VCPUs to update *dom_lvt*. The *dom_lvt* variable is used periodically to coercively increase the LVT of lagging VCPUs to keep all the VCPUs of the DOM in sync. Note that the *nw_dom* data-structure does not need the *weight* and *cap* variables of the CSX.

```
struct csched_private {
   spinlock_t lock;
   struct list_head active_sdom;
   uint32_t ncpus;
   struct timer  master_ticker;
   unsigned int master;
   cpumask_t idlers;
   uint32_t weight;
   uint32_t credit;
   int credit_balance;
   uint32_t runq_sort;
};
```

```
struct nw_private {
   spinlock_t lock;
   struct list_head active_sdom;
   uint32_t ncpus;
   struct timer  master_all_lvt_ticker;
   struct timer  master_lvt_ticker;
   struct timer  master_ticker;
   unsigned int master;
   cpumask_t idlers;
   s_time_t sys_lvt;
};
```

Figure 7: CSX and NSX private data

As mentioned previously, the scheduler contains an *nw_priv* object of type *nw_private*. It contains a queue named *active_sdom*. The *ncpus* variable gives the number of cores supported by the underlying hardware. Importantly the scheduler object contains a time variable named *sys_lvt* that keeps track of the global *LVT* (the minima of all *dom_lvts* across all DOMs except for DOM0 and DOM 32767). The *sys_lvt* is equivalent to the traditional concept in simulators of *now* in simulation time. The *sys_lvt_next* holds the maxima of all *dom_lvts.*

The *nw_priv* maintains three timers. The first is the *master_ticker* used to update the *dom_lvt* across all DOMs. The second is the *master_lvt_ticker* used to synchronize the VCPUs corresponding to a DOM to their *dom_lvt*, thus artificially advancing the LVTs of the lagging VCPUs in the DOM to *dom_lvt*. The third timer named *master_all_lvt_ticker* synchronizes all LVTs of all VCPUs across all the DOMs to *sys_lvt_next*, thus advancing the LVTs among all the VCPUs to the higher value.

The variable named *idlers* keeps track of the idling *VCPUs* and the variable *lock* is used for updating this global object. Figure 7 compares the global data-structure that is *csched_private* of CSX and *nw_private* of NSX. Most of the CSX variables namely, *credit*, *weight*, *balance* and *runq_sort* of the *csched_private* data-structure are removed, while keeping the remaining others unchanged.

C. *Implementation of Features*

To fulfill simulation time ordering requirement, each VCPU keeps track of its utilized ticks (time units) of its execution using variables *nticks* and *ref_time*, the reference time from which *nticks* has elapsed. These two values are used to calculate the LVT of the VCPU.

The utilization of the VCPUs increases as the interconnected DOMs execute any application and as they do, the corresponding VCPU tick values increments. Hence the simulation time increases as it is dependent on the LVT values of VCPUs. The least value of the VCPU can be safely considered as the elapsed simulation time in a single core machine. In a multi-core system, we consider the maximum LVT among the VCPUs of a DOM as the *dom_lvt* and the minimum of the *dom_lvt*s is considered as the simulation time.

A periodic timer named *ticker* corresponding to each core (in a multi-core processor) is maintained in the *nw_pcpu* data-structure. This periodic timer goes off at every *tick* and when this goes off, the accounting (charging tick time) is performed to the *current* runnable VCPU. Then a soft-interrupt for scheduling this VCPU is raised on the physical core, which is selected either based on the provided VCPU-PCPU affinity-map or on the criteria that the VCPU has most idling neighbors in its grouping. This code in NSX is exactly same as that found in CSX. Thus at every tick, generated by the *ticker* timer from either of the nw_*pcpu* object corresponding to a cpu-cores a VCPU is scheduled to run on a CPU-core.

Another global timer named *master_ticker* is made to go off for every 10 ticks. This calculates the LVT of the DOM, which is the maximum of the LVT values among all its VCPUs. This is carried out for every DOM and in each case the *dom_lvt* is recorded in its corresponding *nw_dom* object. While this doesn't have much functional significance, it allows us to achieve better performance and it is also used to artificially increment the LVT of the VCPU with lagging LVT to its *dom_lvt* value, periodically, as discussed next.

Accounting for the effect of idle-VCPU cycles

If left unchecked the VCPUs run asynchronously. Even, within a single DOM, there could be differences in the LVT values of its VCPUs, as this is dependent on the task assignment of the guest OS onto its VCPUs. To limit these differences, and their subsequent propagation, between the LVT values of the VCPUs within a single DOM, we adjust the LVT values of all the VCPUs to the maximum of the LVT values among them (i.e. *dom_lvt*). This adjustment is carried

out at regular intervals, which we chose to be a minute. A periodic timer *master_lvt_ticker* is used for this purpose. When this timer goes off, the *Adjust_lvt* routine is called and this function adjusts the LVTs of all the VCPUs to the *dom_lvt*.

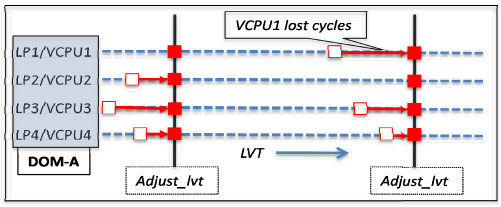


Figure 8: Accounting for VCPUs with lagging LVTs

The NSX scheduler enables us to capture the effect of lost cycles due to the idle cycling of the peer-VCPU cores while using the multi-core processors. Note that the idle VCPUs are not charged and hence their LVT value remains stationary, while the LVTs of the active VCPUs increase. This results in a staggering of LVT values for different VCPUs within the same DOM as shown in the Figure 8. Since the VCPUs being idle or active is dependent on the guest OS's scheduling mechanism, the lagging VPCUs can be tracked with periodic updates. To illustrate, Figure 8 shows a DOM with 4 cores and the LVT values of each VCPUs staggered and being pulled to the *dom_lvt* value during periodic updates serviced by the *Adjust_lvt* routine.

Accounting the effect of Idle Doms in network

Similarly, a *master_all_lvt_ticker* is used to adjust the LVTs of the DOMs (i.e. all the VCPUs) to the *sys_lvt_next* value after some long time, as shown in Figure 9. The only difference is that the *Adjust_all_lvt* routine that services this ticker checks if any of the DOMs are lagging beyond certain difference period (*NW_MAX_DIFF*) from *sys_lvt* before pushing the LVTs of all the VCPUs from all DOMs to *sys_lvt_next*. In our runs the *NW_MAX_DIFF* was set to 10 ticks.

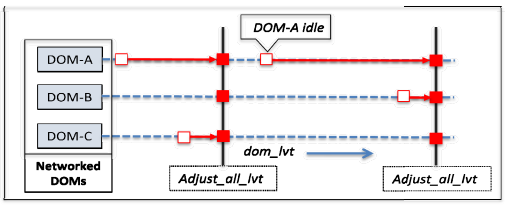


Figure 9: Accounting for Idle DOM time

This updating procedure of LVTs of all VCPUs to *sys_lvt_next* accounts for the time of idle-DOMs during the network simulation. All global timers are maintained in *nw_priv*. Thus the LVT corresponding to each VCPU, along with the help of intermediate VCPU-specific timers and global timers, keep track of the simulation time.

The VCPU with the least LVT must always be scheduled for next execution. This is ensured during the insertion of VCPU in to the run-queue, which happens either during *nw_schedule* is called or when the VCPU awakes from sleep (*nw_wake*). Since, the LVT value of VCPUs is always non-decreasing and never changes in between updates, it is not necessary to sort the queue in between as required in the CSX. However, this criterion is not sufficient to ensure the required property of "least-LVT scheduled first" on machines with multi-core processors, and, as such, the scheduling needs to be extended further, as described next.

The scheduler maintains a run queue for every physical core. When the *nw_load_balance* routine is used, it safely steals the least LVT valued VCPU from its peer PCPU's *nw_pcpu's runq* and schedules it for execution. This functionality is very similar to that carried out by the CSX. The only difference is that CSX looks for a VCPU with higher-priority value while the NSX looks for the least-LVT value. This should ensure the least-LVT valued VCPU is always scheduled first.

New DOM Boot Process Considerations

Not all the DOMs participating in the simulation can be expected to start at the same time. Whenever a new DOM needs to be added to the existing simulation scenario, it needs to start at the current simulation time, so that its execution is in coherence with other DOMs. This is ensured in the *vcpu_init* routine, which assigns the *sys_lvt* value to all the VCPUs of the newly active DOM. Thus ensuring that the arbitrarily added DOM works in coherence with the other DOMs participating in the simulation.

Also, since the *sys_lvt* computations are carried out only across the active DOMs, we ensure that a sudden departure of any DOM in the middle of the simulation would not affect the rest of the simulation. These features provide an opportunity to arbitrarily grow or shrink the number of DOMs simulated during the simulation.

Scheduling DOM0

The DOM0 is not only an interface for interacting and monitoring the other DOMs but also the one where a link (bridge or a router) exists that interconnects the user DOMs. In our example network used for testing, we create a bridge in DOM0 and all user DOMs are connected to each other via this bridge. Consequently, all the communications between the user DOMs passes through DOM0. Hence, sufficient number of compute cycles must be provided to DOM0, so that the user interactivity doesn't suffer and also other necessary tasks for the functioning of the communicating DOMs happen without delay. In a similar way, we also need to ensure that the DOM0 accedes to other DOMs, so that they are not starved.

To achieve this, the VCPU's of DOM0 are always maintained at *sys_lvt* of the DOMs. Note that *sys_lvt* is the minimum of *dom_lvt* and the *dom_lvt* is the maximum of all the VCPUs LVT values in that DOM. This essentially says that the DOM0 VCPUs accede to the VCPU with the least LVT, if there is one that lags behind the *sys_lvt* value. The updates to *dom_lvt* and *sys_lvt* are performed periodically after every '*t*' ticks. By increasing or decreasing this tick value, we were able to vary the number of CPU-cycles provided to DOM0. In our testing we assigned *t =10* ticks.

Small Time-slice

The credit scheduler of Xen uses a default time-slice value of 30ms, which is much larger than the network link latencies

required in our simulation scenarios. Hence we made modifications to CSX so that the time slice can be reduced much below 30ms in our experiments.

Every time Xen needs to schedule a new VCPU, it invokes the *nw_schedule* routine. This routine picks up the least-LVT valued VCPU to execute and provides a *time-slice* (number of *ticks*) for the VCPU to execute. Xen measures time slices in milliseconds, which we changed to microseconds for finer time granularity control. The number of ticks in a *time-slice* was kept 1. Thus, by changing the tick size in the pre-processor statements of the scheduler code, we were able to alter the tick size as needed.

D. *Overall Scheduling Process in NSX*

Schedule service routine: Xen's scheduling framework ensures that there is always at least one VCPU in the *run-queues* of each PCPU. When the scheduler is interrupted, the service routine in NSX inserts the *current* (currently being serviced VCPU) into the run-queue of the PCPU (cpu-core) that was interrupted. Then the VCPU with least LVT is searched in all the *run_queues* of the existing PCPUs (cpu-cores). This VCPU is removed from the *run_queue* and would be scheduled to run next. The time-slice (number of ticks) that this VCPU should run is also provided in this routine. The selected VCPU and the *time-slice* are returned back from this service routine of NSX to Xen. Xen, during context switching process, assigns the selected VCPU as *current* and the *current* VCPU is executed on the actual physical core.

VCPU accounting: The *master_ticker*, that goes off periodically at every *tick* (same as *time-slice* in our implementation) increments the LVT of the *current* VCPU by *tick* size. Then it raises a *schedule* software interrupt on the core that has the most idling peer VCPUs in its DOM. This *schedule* interrupt is serviced by the schedule service routine.

IV. EXPERIMENTAL SETUP

A. *Test Scenario*

We implemented a simple parallel program using the Message Passing Interface (MPI) library to test and also to compare NSX against CSX. The parallel execution is distributed across the guest DOMs such that exactly one process of the parallel execution is run on one guest DOM. Three domains, DOM-A, DOM-B and DOM-C, hosts three MPI processes of rank-0, rank-1 and rank-2, respectively, which participate in multiple *messaging rounds*. In a *messaging round*, the rank-0 process (in DOM-A) sends a message to rank-1 process (in DOM-B) and a then message to rank-2 (in DOM-C). The rank-1 process that was waiting for message from the rank-0 process receives it and then sends out a message to rank-2 process immediately. The message sent out by rank-0 is an integer 0 and the message sent out by rank-1 is an integer 1. We use *MPI_ANY_TAG* and *MPI_ANY_SOURCE* while receiving an MPI messages to ensure that the MPI_Recv permits reception in any order for incoming messages.

Note that the interacting parallel processes are running on different DOMs. Hence for an individual messaging round, if we were to consider the network delay experienced by the communication message a constant, and, if the DOMs are being scheduled in the strict time-order, then the causal order of arrival of messages received by rank-2 (in DOM-C) must be 0-1. In other words, rank-2 must first receive the message from rank-0, before it receives the message from rank-1. Further, the blocking send and receive MPI calls used in realizing the test-program largely reduces the time gap between the direct message from rank-0 and the relayed message from rank 1, at rank-2. Hence, this makes the test fine grained and tight.

If we were to run multiple *messaging rounds* one after another, then the correct time-ordered messaging sequence received by rank-2 will be a sequence of [0-1-0-1-0-1…]. An occurrence of a break in this ordering is counted as a single breach in time order or a *unit error*. We count all such errors committed in a large number of *messaging rounds* to determine the *error percentage* from a single run. The mean or average of several of these runs are used to obtain a *mean-error-percentage* value. In all of our experiments, the number of *messaging rounds* in each run were 1000, and the *mean-error-percentage* value was obtained from averaging the *error-percentage* over 30 runs.

To get the elapsed (wall clock) *run time* of the experimental runs, the *real-time* of the parallel execution is recorded on the DOM on which it was initiated (DOM-A). The *run time* was recorded for each of the 30 runs and their mean value is plotted.

For the CSX readings, the *weights* for all the DOMs were kept same (at 256) and none of them was capped, thus ensuring maximum fairness in the working of CSX.

B. *Test Setup*

The test setup comprised a Xen hypervisor v3.4.2, with Linux running as its DOM0 and all the other three guest-DOMs (DOM-A, DOM-B and DOM-C). Each of the guest-DOMs was assigned a static IP address.

In DOM0, a bridge named *privatebr*, for establishing network connections between the guest-DOMs was created using the *brctl* tool. The DOM0 itself does not connect to *privatebr* and hence remains separate from the network of guest-DOMs. However, since the *privatebr* exists in DOM0, all the communication messages from the guest-DOMs pass through DOM0.

C. *Test Machine*

The setup and the test runs were carried out on a MacBook-Pro with Intel® Core™ 2 CPU T7600 @ 2.33 GHz, with 3 GB memory, running OpenSUSE 11.1 Linux over Xen 3.3.1.

The source code of Xen 3.4.2 distribution was used for making scheduler modifications. The resulting boot image xen-3.4.2.gz was used to boot the hypervisor, and the boot-loader grub configuration was changed to enable the selection of modified Xen during the boot up. All the guest-DOMs were configured to run OpenSUSE 11.1 Linux and they were installed as para-virtualized DOMs for performance reasons.

The test program was written in the C programming

language using MPI (OpenMPI v1.4.1) library.

V. PERFORMANCE RESULTS

By default, CSX maintains a tick size of 10ms, and during scheduling, every VCPU is allotted a time-slice equal to thrice the tick size. Hence in our experiments the maximum tick size is set to 30ms and the minimum is 30µs. Below 30µs the interactivity suffers heavily (for both CSX and NSX) and the performance of the guest DOMs deteriorates because of high context switching rate. Performance is tested in two major case scenarios. In the first set, each DOM is configured to have single VCPU, whereas, in the second case, each DOM contains two VCPUs.

A. *Case 1: Single VCPU per DOM*

Thirty runs with each run performing 1000 *messaging rounds* were carried out. The mean-percent-error and runtime are shown in the Figure 10 and Figure 11. A table of 95% confidence intervals (CI) is provided for CSX and NSX for the top two curves.

For NSX, we see a dramatic reduction in the *mean-error* with the increase in the tick-size. In the CSX the *mean-error* drops by a small amount initially and steeply increases later with the increase in the tick size. From Figure 10, we see that as the tick size increases from 30µs to 30ms, the mean error percentage decreases from 4% to 0%, whereas CSX increases from 3% to 56%.

Figure 11 shows almost same runtime for both NSX and CSX. However, the runtime of NSX suffers at lower tick sizes. The increase in the runtime with increase in the tick size is expected because larger than necessary tick-size is allotted for every VCPU, which essentially is wasted.

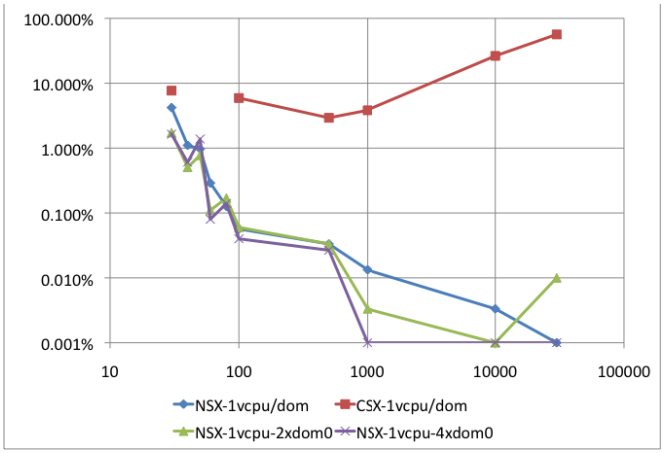
B. *Case 2: Multiple VCPUs per DOM*

A similar reduction in the *mean-error* is observed in Figure 12. However, unlike in the 1-VCPU scenario, it does not go down to absolute zero, but stays around 0.04%. Similar to the 1-VCPU scenario, an increase in CSX's mean-error with increase in the tick size is observed. At 30ms tick-size, the error was at 49%.

The runtime plot in Figure 13 shows a steep increase in the NSX runtime with increasing tick size, which is decreased by increasing the DOM0 tick size, as discussed shortly.

C. *Analysis of Results*

Causal error in CSX: The CSX caps the *over-scheduled* VCPUs and chooses *under-scheduled* VCPU for scheduling. In doing so, it constantly re-orders the *run_queue* based on priority, and, when all VCPUs become over-scheduled, the credits are re-assigned. This process continues over the simulation.



	CSX-1VCPU 95% CI			NSX-1VCPU 95% CI		
Tick size µs	Lower limit	Mean error	Upper limit	Lower limit	Mean error	Upper limit
30	5.31%	7.68%	10.06%	3.07%	4.21%	5.35%
100	4.75%	5.92%	7.09%	0.03%	0.06%	0.08%
500	2.23%	2.93%	3.63%	0.02%	0.03%	0.05%
1000	2.63%	3.83%	5.03%	0.00%	0.01%	0.03%
10000	22.11%	26.37%	30.62%	0.00%	0.00%	0.01%
30000	50.04%	56.51%	62.97%	0.00%	0.00%	0.00%

Figure 10: Mean-error (y-axis in %) vs. tick-size (x-axis in microseconds) for the NSX, the CSX, the NSX with 2x and 4x DOM0 tick sizes. Mean error is obtained from 30 runs with each run of 1000 messaging rounds in a 1-VCPU per DOM scenario

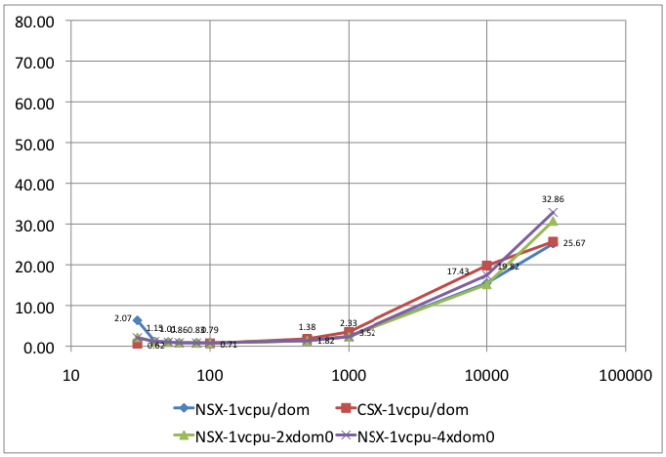
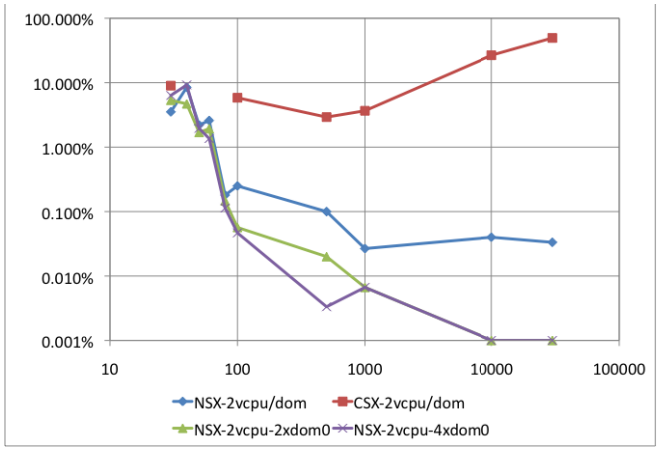


Figure 11: Single VCPU/DOM case - runtime curves of *mean-error* runs, for the NSX, the CSX, the NSX with 2x and 4x DOM0 tick size

The CSX maintains fairness (in a non-simulation sense) but causality is affected. This is because the program execution and communications are asynchronous and, the load on the processes in the parallel program is usually not equal. Capping one VCPU while scheduling others increases the probability of committing causal error. For example, capping the VCPUs of DOM-A holds back DOM-A from sending message to DOM-C in time as expected; the under-scheduled DOM-B VCPU, given more-cycles, will be able to send message to DOM-C before DOM-A does, hence creating a causal error.



	CSX-2VCPU 95% CI			NSX-2VCPU 95% CI		
Tick size µs	Lower limit	Mean error	Upper limit	Lower limit	Mean error	Upper limit
30	6.31%	9.00%	11.68%	2.26%	3.52%	4.78%
100	4.40%	5.84%	7.27%	0.19%	0.25%	0.31%
500	2.04%	2.92%	3.80%	-0.02%	0.10%	0.22%
1000	2.52%	3.67%	4.82%	0.01%	0.03%	0.04%
10000	22.81%	26.63%	30.46%	0.02%	0.04%	0.06%
30000	43.60%	49.43%	55.25%	0.02%	0.03%	0.05%

Figure 12: Tick size vs. error plots similar to Figure 10, but with 2 VCPUs per DOM

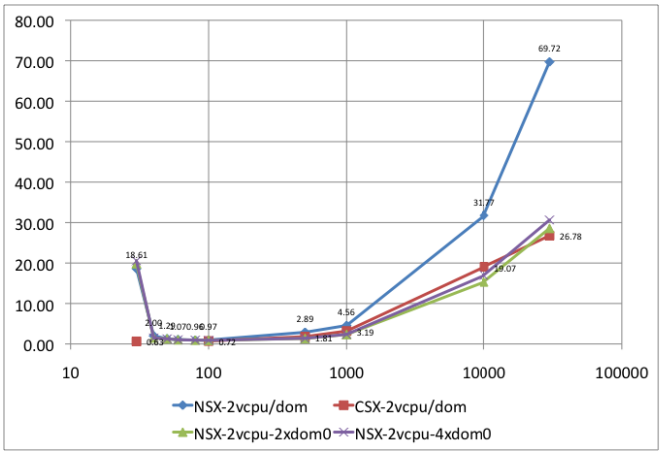


Figure 13: Tick size vs. runtime similar to Figure 11, but with 2 VCPUs per DOM

Causal error in NSX: In NSX, VCPU's cycles are always provided and are not blocked anytime. However, the VCPU with least-LVT in the run-queue will be executed first. This results a staggered time line for each VCPU's LVT, but they are adjusted periodically as discussed earlier.

As seen in 1-VCPU and 2-VCPU case studies, the incidence of time-order error occurrences are at lower tick-sizes. The error occurrences tend to increase as the tick size decreases. The high-frequency context switching between the VCPUs is responsible for the errors. This reasoning is supported by the runtime plots showing higher runtime at lower tick sizes for executing the same set of experimental scenarios.

Since DOM0 manages the bank-end drivers and the network-bridge, and also serves as interaction interface, its starvation due to a higher context switching frequency aggravates the error-rate. The error-rate can be significantly alleviated as seen in Figure 10 and Figure 12, by providing larger tick sizes to the DOM0. In our experiments, we provided twice (2x) and four times (4x) tick sizes to DOM0. This also reduces the runtime as seen in Figure 11 and in Figure 13, and, the impact of this change is clearly evident. As the DOM0's LVT does not affect the *sys_lvt* (the simulation time), this change does not adversely impact the error in the test-scenario.

Mean-error and Runtime at lower tick sizes: The smallest-tick size is very essential to simulate low-latency and high-bandwidth network. Hence, it is interesting to detect the earliest point at which a continued reduction in tick size gives acceptable error without significant increase in the runtime. At 20µs tick size, both CSX and NSX pose interactivity problem. Thus the least tick-size with which we could run the experiments was 30µs.

For the 1-VCPU per DOM scenario, Figure 10 and Figure 11, with 2x DOM0 indicate the best error reduction and at 60µs, the error reduces to 0.1% and the runtime 0.86 seconds. From Figure 12 and Figure 13, we see that, in the 2 VCPU/DOM scenario, the 2x DOM0 case gives an overall best error reduction and a better run time. At 80µs, the error reduces to 0.18% and the run time by 0.95 seconds.

VI. SUMMARY

The following table summarizes the results.

Simulation-time Global time Idle-VCPU accounting Idle-DOM accounting	 Achieved by *sys_lvt*. Is done periodically. Is done periodically.
Causality Time-order Errors	 Ensured in scheduling Errors reduced to <1%
New-DOM needs DOM add/remove No detrimental effect Scenario modification reflected in simulation	 Supported Works as expected VCPU of a new DOM initialized to *sys_lvt* ensures addition
DOM0 needs Enough CPU-cycles Lack of CPU-cycles to DOM0 might introduce causal errors Starvation of DOMs by DOM0	DOM0 is always maintained at sys_lvt Is addressed by changing the tick-size just for DOM0 Absence of starvation implicit from results.
Small tick slice Smallest tick size Error less than 1% Achievable runtime for required error bound	1VCPU/DOM- 60µs 2VCPU/DOM- 80µs 1VCPU/DOM- 0.1% 2VCPU/DOM- 0.18% 1VCPU/DOM- 0.86s 2VCPU/DOM- 0.95s

VII. RELATED WORK

Within the area of network simulation research, there have been several prominent advancements realized in the form of simulators and testbeds such as Emulab, GTNetS, PDNS, SSF, DaSSF, PRIME, and Qualnet, to name just a few. Among the systems that employed virtualization for higher fidelity, the well-known ones include time dilation [1], time jails[9],

virtual time[10], real-time interface optimizations[8,11,12], and others [13-16]. However, we are not aware of prior work on simulation-specific schedulers for simulating multi-core guests on multi-core hosts. It may be possible to integrate our scheduler into one of the current simulation frameworks such as Qualnet, NS3 or PRIME, which we intend to explore as future work.

VIII. CONCLUSION AND FUTURE WORK

In the evolution of network simulators with higher fidelity, virtual machine-based behaviors clearly play an important role in the future. This entails supporting multi-core guest VMs and multi-core host nodes executed in simulation time order to control VM interaction within a host node. Traditional fairness-oriented (or utilization-oriented) VM schedulers are intuitively inappropriate for simulation needs. Our experiments corroborate this mismatch empirically, showing the need for simulations-specific schedulers. To address the gap, we developed a new simulation-focused scheduler called NetWarp scheduler in our NetWarp simulation system for multi-core VMs simulated using multi-core hosts. We realized a working implementation within the Xen hypervisor framework. Preliminary experimental results from our experiments have shown here that simulation-oriented schedulers such as ours can indeed dramatically reduce time-ordering inaccuracies while still incurring minimal runtime overhead compared to non-simulation schedulers optimized for processor utilization. Additional research is needed to evaluate on larger platforms. Host nodes with dozens of cores are now available. We are presently exploring this approach on a host with 24 cores (four hex-core sockets), and hope to have performance data on this platform in the near future.

ACKNOWLEDGEMENTS

This paper has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the U.S. Department of Energy. Accordingly, the United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

REFERENCES

[1] D. Gupta, et al. "To Infinity and Beyond: Time-Warped Network Emulation," in Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI'06), pp. 87-100, San Jose, CA, USA, 2006.

[2] Jenna N. Mathews et al., "Running Xen, A Hands-On Guide to the Art of Virtualization," ISBN 978-0-132-34966-6, Prentice Hall, 2008.

[3] http://www.xen.org/products/xenhyp.html.

[4] QEMU: http://wiki.qemu.org/Main_Page

[5] David Chisnall, "The Definitive Guide to the Xen Hypervisor," ISBN 978-013-234971-0, Prentice Hall, 2008.

[6] "Xen Wiki: Credit-Based CPU Scheduler," http://wiki.xensource.com/xenwiki/CreditScheduler.

[7] http://book.xen.prgmr.com/mediawiki/index.php/Scheduling

[8] Jason Liu, Yue Li and Ying He, "A Large Scale Real-time Network Simulation Study Using PRIME," in Proceedings of the Winter Simulation Conference (WSC), 2009.

[9] A. Grau, et al., "Time Jails: A Hybrid Approach to Scalable Network Emulation," in Proceedings of the 22nd Workshop on Principles of Advanced and Distributed Simulation (PADS) , pp. 7-14, Rome, Italy, 2008.

[10] Y. Li, et al., "Towards Scalable Routing Experiments with Real-time Network Simulation," in 22nd International Workshop on Principles of Advanced and Distributed Simulation (PADS) , pp. 23-30, Rome, Italy, 2008.

[11] D. Gupta, et al., "DieCast: Testing Distributed Systems with an Accurate Scale Model," in Proceedings of the 5th ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI), San Francisco, CA, 2008.

[12] Elias Weingrtner, et al., "Synchronized Network Emulation: Matching Prototypes with Complex Simulations," SIGMETRICS Performormance Evaluation Review, vol. 36, pp. 58-63, 2008.

[13] S. Maier, et al., "Scalable Network Emulation: A Comparison of Virtual Routing and Virtual Machines," in IEEE Symposium on Computers and Communications, pp. 395-402, Aveiro, Portugal, 2007.

[14] C. Bergstrom, et al., "The Distributed Open Network Emulator: Using Relativistic Time for Distributed Scalable Simulation," in Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation (PADS), 2006.

[15] G. Apostolopoulos and C. Hasapis, "V-eM: A Cluster of Virtual Machines for Robust, Detailed and High-performance Network Emulation," in Proceedings of the 14th IEEE International Symposium on Modeling, Analysis and Simulation of Computing and Telecommunication Systems, pp. 117-126, Monterey, CA, USA, 2006.

[16] C. Kiddle, et al., "Improving Scalability of Network Emulation through Parallelism and Abstraction," in Proceedings of the 38th Annual Simulation Symposium, pp. 119-129, 2005.

PrimoGENI: Integrating Real-Time Network Simulation and Emulation in GENI

Nathanael Van Vorst, Miguel Erazo, and Jason Liu
School of Computing and Information Sciences
Florida International University
Emails: {*nvanv001, meraz001, liux*}*@cis.fiu.edu*

***Abstract*—The Global Environment for Network Innovations (GENI) is a community-driven research and development effort to build a collaborative and exploratory network experimentation platform—a "virtual laboratory" for the design, implementation and evaluation of future networks. The PrimoGENI project enables real-time network simulation by extending an existing network simulator to become part of the GENI federation to support large-scale experiments involving physical, simulated and emulated network entities. In this paper, we describe a novel design of PrimoGENI, which aims at supporting realistic, scalable, and flexible network experiments with real-time simulation and emulation capabilities. We present a flexible emulation infrastructure that allows both remote client machines and local cluster nodes running virtual machines to seamlessly interoperate with the simulated network running within a designated "slice" of resources. We show the results of our preliminary validation and performance studies to demonstrate the capabilities and limitations of our approach.**

***Index Terms*—network simulation, emulation, real-time simulation, virtualization**

I. INTRODUCTION

The success of Internet can be attributed to many factors. However, many argue that the very success has brought the ossification of its own design that makes it difficult to sometimes realize even the simplest changes. Political and economic factors notwithstanding, the success of transforming academic research to empirical implementation depends on the availability and trustworthiness of network testbeds to correctly expose important design and operational issues.

Considerable progress has been made in recent years in network testbed design to foster network innovations. The GENI initiative [1] capitalizes on the success of many previous efforts and strives to create a single collaborative and exploratory platform for future Internet research, bringing together the latest advances of network testbed research. The GENI design is built on four premises [2]:

1) *Programmability:* the ability to execute custom software on GENI-enabled facilities deep inside the network hierarchy;
2) *Resource sharing*: sharable resources among multiple users and experiments, possibly through virtualization;
3) *Federation*: interoperability among various resources provided by different organizations; and
4) *Slice-based experimentation*: supporting network experiments running independently within "slices" (a subset of GENI resources).

While GENI offers a cross-platform virtualization solution for physical and emulation testbeds, simulation is ostensibly missing in its original design. Although physical and emulation testbeds provide the necessary realism in prototyping and testing new network designs, simulation is more effective for studying complex behaviors of large-scale systems, which are both difficult to handle using realistic settings and intractable to close-form mathematical and analytical solutions.

PrimoGENI is a project that aims to incorporate real-time network simulation capabilities into the GENI "ecosystem". Real-time network simulation refers to simulation of potentially large-scale networks in real time so that the virtual network can interact with real implementations of network protocols, network services, and distributed applications. PrimoGENI uses our existing real-time network simulator, called PRIME [3], which is a real-time extension to the SSFNet simulator, capable of running on parallel and distributed machines for large-scale network simulations [4] . We augment PRIME with the GENI aggregate API, through which the experimenters can allocate resources and run experiments. PRIME can interoperate with the physical, simulated and emulated network components. In particular, network applications or protocols can run unmodified on emulated hosts, which are implemented as virtual machines, and interact with the simulated network. Network traffic generated by these applications or protocols is "conducted" on the simulated network operating in real time, with appropriate packet delays and losses calculated according to the simulated network conditions. In this respect, the simulated network is considered indistinguishable from a physical network.

In this paper, we present the design of PrimoGENI, which uses a current GENI control framework and extends the existing emulation infrastructure to support simulation and emulation experiments on a compute cluster. We highlight a set of major design decisions that direct our development and prototyping activities with the objective of substantially broadening GENI's capability in supporting realistic, scalable, and flexible experimental networking studies. Our contribution is three-fold:

1) PrimoGENI is the first of its kind to provide an end-to-end solution for building a real-time network simulation testbed that allows users to construct, deploy, and run network experiments involving physical, simulated and emulated network components in GENI.

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

2) PrimoGENI features an infrastructure that allows multiple network experiments to run independently within their own slices, each possibly consisting of remote client machines and local cluster nodes running virtual machines that seamlessly interoperate with the simulated network.
3) We conducted preliminary validation and performance studies to demonstrate the capabilities and limitations of our approach.

The rest of the paper is organized as follows. In Section II we present the background and describe related work and the current GENI architecture. We describe in detail the PrimoGENI design in Section III. We present the result of our validation and performance studies of our preliminary implementation in Section IV. Finally we conclude this paper with a discussion of our ongoing and future work in Section V.

II. BACKGROUND

Network testbeds are commonly used for prototyping, evaluating, and analyzing new network designs and services. *Physical testbeds* (such as WAIL [5] and PlanetLab [6]) provide an iconic version of the network for experimental studies, sometimes even with live traffic. They provide a realistic testing environment for network applications, but with limited user controllability. Further, being shared facilities, they are overloaded due to heavy use, in which case it can severely affect their availability and accuracy [7]. An *emulation testbed* can be built on a variety of computing platforms, including dedicated compute clusters (such as ModelNet [8] and EmuLab [9]), distributed platforms (such as VINI [10]), and special programmable devices (such as ORL [11] and ORBIT [12]). While most emulation testbeds provide basic traffic "shaping" capabilities, *simulation testbeds* can generally achieve better flexibility and controllability. For example, ns-2 [13] features a rich collection of network algorithms and protocols that can be selected to model a myriad of network environments, wired or wireless. Simulation can also be scaled up to handle large-scale networks (e.g., SSFNet [4], GTNeTS [14] and ROSSNet [15]). It would be difficult and costly to build a large-scale physical or emulation testbed.

Real-time simulation is the technique of running network simulation in real time and thus can interact with real implementation of network applications [16]. Most existing real-time simulators (e.g., [17]–[20]) are based on existing network simulators extended with emulation capabilities. PRIME is a discrete-event simulator designed to run on parallel and distributed platforms and handle large-scale network models. PrimoGENI uses PRIME for real-time simulation, along with several important features:

- PRIME has an emulation infrastructure based on VPN that allows distributed client machines to remotely connect to the real-time simulator [21]. In a previous study, we conducted large-scale peer-to-peer content distribution network experiments using this emulation infrastructure [22].
- PRIME provides 14 TCP congestion control algorithms (mostly ported from the Linux TCP implementation), which have been validated carefully through extensive simulation and emulation studies [23].
- To allow large-scale traffic and reduce the computational requirement, PRIME applies multi-scale modeling techniques, allowing fluid traffic models to integrate with emulation traffic and achieving more than three orders of magnitude in performance over the traditional packet-oriented simulation [24].

GENI is a network research infrastructure that combines several prominent network testbeds, such as EMULAB [9], PlanetLab [6], and ORBIT [12], as a single collaborative and exploratory platform for implementing and testing new network designs and technologies. The GENI design consists of three main types of entities: clearinghouses, aggregates, and principals. A *clearinghouse* provides the central management of GENI resources for experimenters and administrators; specifically, it provides registry services for principals, slices and aggregates, and authentication services for accessing the resources. An *aggregate* represents a group of components encapsulating the GENI sharable resources (including computation, communication, measurement, and storage resources). When an experimenter from a research organization (i.e., a *principal*) decides to conduct a GENI experiment, she negotiates with the clearinghouse and the associated aggregate managers through an elaborate resource discovery and allocation process. In response to the experimenter's request, each participating aggregate allocates the requested resources for the experimenter, which are called *slivers*. Jointly, these slivers from all participating aggregates form a *slice*, which represents the set of allocated resources in which the experimenter will run network experiments.

A experimenter acquires resources from the GENI aggregates using a *control framework* run by the clearinghouse. GENI supports several control frameworks, including ProtoGENI [25], PlanetLab [6], ORBIT [12], and ORCA [26]. These control frameworks implement the GENI aggregate manager API that allows the experimenters with proper authentication to query for available sources, allocate resources upon the resource specification of an experiment, set up the resources for the experiment, and reclaim the resources after the experiment has finished. In particular, PrimoGENI extends the ProtoGENI control framework to manage, control and access the underlying resources. ProtoGENI is developed based on EmuLab [9], which is an emulation testbed implemented on a compute cluster. ProtoGENI extends EmuLab with key GENI functions, such as the registry services for principals, slices, and aggregate/component managers.

III. THE PRIMOGENI APPROACH

PrimoGENI runs a network model in real time, and thus can interact with real applications. Real-time simulation can provide accurate results since it is able to capture detailed packet-level transactions. It is flexible since one can easily explore the simulation model to answer what-if questions.

It also supports large-scale network experiments potentially with millions of simulated entities (hosts, routers, and links) and thousands of emulated hosts running unmodified network protocols and applications.

To interoperate with other GENI facilities, PrimoGENI functions as a GENI aggregate; as such, the experimenters can use the well-defined API to remotely control and realize network experiments consisting of physical, simulated and emulated network entities exchanging real network traffic.

A. Architecture

PrimoGENI provides an integrated development environment (IDE), called *slingshot*, for the experimenters to manage the "life cycle" of a network experiment, which includes model construction, configuration, resource specification, deployment, execution, monitoring and control, data collection, visualization and analysis. Slingshot provides a Python console for constructing and configuring network models interactively. One can also specify the network models using Java or XML. A network model describing the target system may consist of the topology of the network and a specification of simulated traffic on the network. One can also designate some network entities as *emulated hosts*, which are either virtual machines or physical machines with specified operating systems to run unmodified network protocols and applications. Slingshot provides a persistent storage for the network models using a database so that they can be inspected, analyzed and possibly reused after the experiment. Slingshot also provides a graphical representation of the network model for the experimenters to conveniently inspect and change the detailed configurations.

Once a network model is determined, the experimenter needs to specify the number of compute nodes to run the network experiment. Slingshot uses the METIS graph partitioner [27] to partition the network model by dividing the network model into subnetworks of approximately equal size and at the same time trying to maximize the communication latencies between the subnetworks (in order to achieve better lookahead for parallel simulation). Emulated hosts are given more weight in the network graph since it is important to spread the virtual machines evenly among the compute nodes as they require more resources. After that, the user can deploy and launch the experiment on a PrimoGENI cluster, which we describe in detail momentarily. During an experiment, the user can use slingshot to monitor the progression of the experiment and, if needed, change the state of the network entities when the experiment is running. The experiment results can be collected and displayed during or after the experiment. So far, we have implemented most of the aforementioned functions, except the online monitoring and control elements, which we leave for future work.

A PrimoGENI cluster consists of a set of compute nodes connected by one or more high-speed switches. PrimoGENI uses the ProtoGENI [25] control framework to manage, control and access the underlying resources. We designate one compute node to run the ProtoGENI aggregate manager and another one to run as the file server. The aggregate manager is responsible for allocating the resources, instantiating and setting up the virtual network, and connecting with the emulated hosts on the virtual machines and physical machines on the PrimoGENI cluster on behalf of the experimenter.

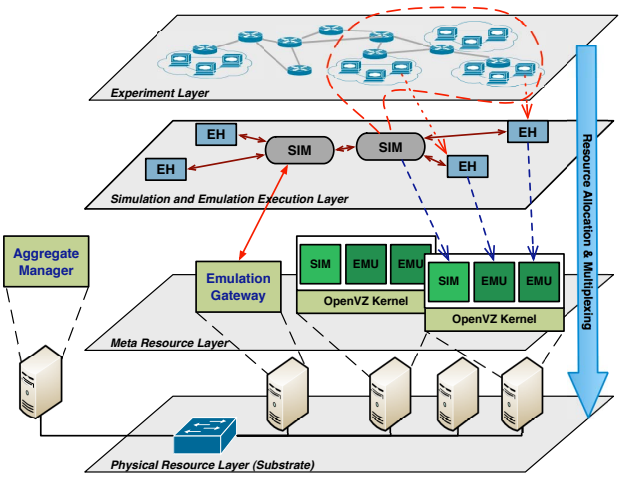


Fig. 1. PrimoGENI aggregate as a layered system

Our design makes a distinction between what we call meta resources and virtual resources. *Meta resources* include compute nodes and network connectivities between the compute nodes (such as a VLAN setup). We call them "meta resources" to separate them from the physical resources (which are known as the substrate). Meta resources also include virtual machines and/or virtual network tunnels. Meta resources are managed by ProtoGENI. PrimoGENI translates the experiment specification into meta resource requirements and instructs ProtoGENI to allocate the resources for the experiment.

Virtual resources are elements of the target virtual network, which include both simulated entities (such as hosts, routers and links), and emulated hosts. Emulated hosts can be implemented as virtual or physical machines, which run unmodified applications, such as web servers or software routers. These emulated hosts communicate with other hosts and routers on the virtual network (either simulated or emulated) according to the network model. Network traffic originated from or destined to an emulated hosts is called *emulated traffic*. All emulated traffic is "conducted" on the virtual network in simulation with proper packet delay and loss calculations as it competes with the simulated traffic.

The PrimoGENI aggregate can be viewed as a layered system, as illustrated in Fig. 1. At the lowest layer is the *physical resources (substrate) layer*, which is composed of compute nodes, switches, and other physical resources that constitute the EmuLab suite. These resources are made known to the clearinghouse(s) and can be queried by researchers during the resource discovery process. A compute node is set up to run the aggregate manager to export the aggregate API to experimenters and clearinghouses. PrimoGENI uses the ProtoGENI control framework, which provides common operations, such as creating, deleting and shutting down slivers. ProtoGENI uses secure socket layer (SSL) for authentication and encryption, and XML-RPC for remote procedure invocation.

The *meta-resource layer* is composed of a subset of resources allocated from the physical resource layer as needed by the experiment. After the requested resources have been successfully allocated by the ProtoGENI control framework, PrimoGENI will bootstrap each compute node by running a customized OpenVZ [28] OS image, which supports virtual machines. Each compute node will automatically start a daemon process, called the *meta-controller*, which awaits commands from slingshot to set up the simulation and emulation execution layer on behalf of the experimenter.

The *simulation and emulation execution layer* is created according to the network model of the experiment. Each compute node serves as a basic scaling unit and implements the operation of a subnetwork or a host that is mapped to it. A scaling unit for a subnetwork consists of a simulator instance, and zero or more emulated hosts each running as a virtual machine. The simulator instances on different scaling units synchronize and communicate with each other using parallel discrete-event simulation techniques. A scaling unit for a host runs the emulated host directly on the physical compute node. This is designed for cases where the emulated hosts have stronger resource requirements and thus cannot be run as virtual machines. PrimoGENI provides an emulation infrastructure to connect the emulated hosts running on virtual or physical machines to the simulator instances that contain the corresponding simulated hosts (see Section III-C). Each compute node runs a meta-controller. We designate one compute node to run the master meta-controller, which relays the commands from slingshot to the other (slave) meta-controllers for setting up the experiment (see Section III-D).

After the experiment starts, the experimenter can interact with the emulated hosts and the simulated network entities on the *experiment layer*. To access the emulated hosts, the experimenter can directly log onto the corresponding computing nodes or virtual machines and execute commands (e.g., pinging any virtual hosts). To access the dynamic state of the simulated network, PrimoGENI will provide an online monitoring and control framework (not yet implemented). This will allow the experimenter to query and visualize the state of the network. In situations where there is a large number of emulated hosts, one can also schedule slingshot commands to be executed at specific emulated hosts.

B. Compute Node Configuration

PrimoGENI uses the ProtoGENI control framework to manage the compute nodes in the cluster; ProtoGENI allows the compute nodes to be loaded with custom operating systems. To scale up emulation experiments, we use virtual machines (VMs) as emulated hosts.

Virtual machines come in different forms, but can be categorized generally as either system-level or OS-level VMs. System-level VMs can run complete guest operating systems (although sometimes with modifications), which can be different from the operating system running on the host machine. System-level VMs can provide strong resource isolation, but can support only limited number of VMs to run simultaneously with the guaranteed resources for each VM. OS-level VMs logically partition the systems into a set of distinct *containers*, each appearing to be a stand-alone machine. OS-level VMs share the same kernel as the host operating system. As such, they do not provide stringent resource isolation, but can support a large number of VMs to run on each physical host.

For PrimoGENI, we chose OpenVZ [28] to run the virtual machines for the emulated hosts where unmodified network applications can run. OpenVZ is an OS-level VM solution that meets our minimum requirement in terms of providing necessary separation of CPU, memory, and network resources among the VMs. CPU and memory separation is needed to limit the interactions of applications running on different VMs. Network separation is necessary to manage network traffic to and from the VMs each with an independent network stack. OpenVZ compartmentalizes the system resources inside the so-called containers; applications can run within these containers as if they were running on dedicated systems. Each container has its own set of processes, file system, and manages its own set of users (including root), and network stack (including network interfaces and routing/forwarding tables). With OpenVZ, PrimoGENI is able to perform network experiments with a large number of emulated hosts.

We preload the compute nodes with OpenVZ in the PrimoGENI cluster and provide an OS template from which the containers will be created. The OS template consists of a root file system and common network applications that may be run on a container (such as `iperf` and `tcpdump`). Additional applications can be added to individual containers using the meta-controller framework, which we describe in Section III-D. Since each container maintains its own file system, and the amount of storage space needed by the file system at each container may range from tens to hundreds of megabytes, which depends on the applications the experimenter wishes run; we would need a lot of space for running network experiments with a large number of emulated hosts. To solve this problem, we choose to use a union file system [29]. A union file system consists of two parts: a read-only file system (RF), which is common to all containers, and a writable file system (WF), which is specific to a container. A copy-on-write policy is used: when a file in RF needs to be modified, the file will be first copied to WF. To prepare the file system for each emulated host, PrimoGENI unions the default OS-template with an initially empty writable base file system. In this way we need only to store changes that later occur for each container as opposed to storing the entire file system.

C. Emulation Infrastructure

The emulation infrastructure in PrimoGENI needs to support high-throughput low-latency data communication between the emulated hosts and the simulation instances. There are two kinds of emulated hosts: collocated and remote. *Collocated emulated hosts* run as virtual machines on the same compute node as the simulator instance that simulates the subnetwork containing the corresponding virtual hosts. *Remote emulated hosts* are either physical compute nodes in the PrimoGENI

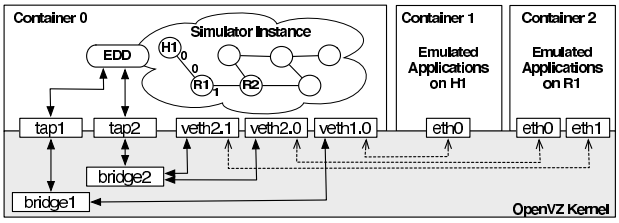


Fig. 2. An interconnection mechanism for collocated emulated hosts

cluster (not VMs), or machines that are not part of the PrimoGENI cluster and potentially at different geographic locations. Remote emulated hosts can run applications which otherwise cannot be run on a virtual machine due to either stringent resource requirements or system compatibility concerns. For example, an emulated host may require a special operating system (or version), or need specialized hardware that is not available on the compute node.

PrimoGENI's emulation infrastructure consists of three major components: the virtual network interfaces, the emulation device drivers, and the interconnection mechanism. The *virtual network interfaces (VNICs)* are installed at the emulated hosts, and treated as regular network devices, through which applications can send and receive network packets. Packets sent to a VNIC are transported to the simulator instance that handles the corresponding virtual host. The simulator subsequently simulates the packets traversing the virtual network as if they originated from the corresponding network interface of virtual host. Upon packets arriving at a network interface of an emulated virtual host in simulation, the packets are sent to the corresponding VNIC at the emulated host, so that its applications can receive the packets.

The *emulation device drivers (EDDs)* are collocated with the simulator instances running on the compute nodes. They are software components used by the real-time simulator to import real network packets sent from VNICs at the emulated hosts. Conversely, EDDs also export simulated packets and send them to the emulated hosts. Our emulation infrastructure supports different types of EDDs for remote or collocated emulated hosts. PRIME provides functions designed specifically for interactive simulations, which include handling real-time events and performing conversions between fully formed network packets and simulation events.

The *interconnection mechanism* connects VNICs on emulated hosts and EDDs with the simulator instances. Its design depends on whether it is dealing with remote or collocated emulated hosts. For remote emulated hosts, PrimoGENI uses the VPN-based emulation infrastructure, which we developed previously in PRIME [21]. In this scheme, OpenVPN clients are run at the the remote emulated hosts, each corresponding to one VNIC. PrimoGENI designates one or more compute nodes to run a modified OpenVPN server, which forwards IP packets to and from the EDDs collocated with the simulator instances. Detailed information about this interconnection mechanism can be found in [21].

Here we describe an interconnection mechanism designed for collocated emulated hosts, which uses Linux bridges and TAP devices. Fig. 2 depicts our design. For each collocated emulated host (an OpenVZ container), we create a software bridge and a TAP device connected to the bridge. Multiple bridges are used to segregate the traffic of each emulated host and ensure the emulated traffic is routed properly. In particular, we do not allow collocated emulated hosts to communicate with each other directly without going through the simulator. There are two possible alternatives. One could use VLANs to separate the traffic, although processing the VLAN headers may introduce additional overhead. One could also use ebtables, which is a mechanism for filtering traffic passing through a Linux bridge [30]. The TAP device is used by the EDD collocated at the simulator instance (in container 0) to send and receive packets to and from the VNICs[1]. The EDD also acts as an ARP proxy responding to the ARP requests from the emulated hosts, so as to direct packets originated from the VNICs to the associated TAP device.

The collocated emulated hosts are running as OpenVZ containers. We use a virtual Ethernet device [31] for each VNIC on the emulated hosts. The virtual Ethernet device is an Ethernet-like device, which actually consists of two Ethernet interfaces—one in container 0 (the privileged domain) and the other in the container where the emulated host is. The two interfaces are connected to each other, so that a packet sending to one interface will appear at the other interface. We connect the interface at container 0 to the bridge which corresponds to the emulated host.

We use an example to show how this mechanism works, by following the path of a ping packet from H1 via R1 and R2 shown in Fig. 2. Suppose both H1 and R1 are emulated in container 1 and 2, respectively. Before container 1 can send a packet to R1, assuming its ARP cache is currently empty, it sends an ARP request out from eth0 asking for the MAC address of R1's network interface eth0. The EDD responds with the MAC address of tap1, which is the TAP device connected to bridge1, the software bridge designated for container 1. Subsequently, container 1 is able to forward the ICMP packet to the EDD, which injects the packet into the simulator (by inserting an event that represents the packet sent out from network interface 0 of the simulated host H1). Once the simulation packet gets to R1 on the simulated network, the packet is exported from the simulator and sent by the EDD to eth0 in container 2 through tap2 and bridge2. Similarly, before container 2 forwards the packet onward to R2, it sends an ARP request out from eth1. The EDD responds with the MAC address of tap2, which is the TAP device connected to bridge2, the software bridge designated for container 2. In this way, the packet can find its way to the simulator, which forwards it on the virtual network.

[1]The latest TAP device implementation prohibits writing IP packets to the kernel, possibly for security reasons. We choose to use a raw socket instead for the EDD to send IP packets.

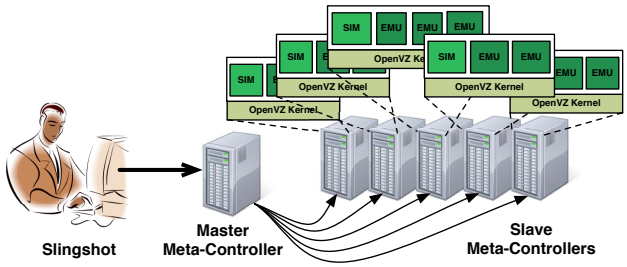


Fig. 3. Meta-controllers for experiment setup

D. Meta-Controllers

As mentioned earlier, PrimoGENI uses the ProtoGENI control framework to allocate the necessary compute nodes to run experiments. After the compute nodes are booted from the customized OpenVZ image, we need a mechanism to configure the compute nodes, create the containers, set up the emulation infrastructure, start the simulation, and launch the applications inside the containers. One can use XML-RPC for this purpose. However, since the commands are originated from slingshot, which is the user interface experimenters use to configure, launch and control the experiments, we would like to have a mechanism that allows the compute nodes to be coordinated locally on the PrimoGENI cluster to achieve better efficiency. In this section, we describe a *tiered* command framework, which we use to set up a network experiment and orchestrate experiment execution.

The command framework, as illustrated in Fig. 3, uses MINA, a Java framework for distributed applications [32]. MINA provides both TCP and UDP-based data transport services with SSL/TLS support. It is designed with an event-driven asynchronous API for high-performance and high-scalability network applications. Our implementation requires each compute node to run a meta-controller daemon process when first booted. When the meta-controller starts, it waits for an incoming connection. At this point the meta-controller takes no role, but after the connection is made, the meta-controller will become either a master or a slave. The difference between a master meta-controller and a slave meta-controller is that the master meta-controller is the one connected with slingshot; commands sent from slingshot, if they are not destined for the master, will be relayed to the corresponding slave meta-controllers. After the compute nodes have been allocated for an experiment, slingshot will choose one compute node to be the master and establish a secure connection to it using the experimenter's key obtained from the aggregate manager. After slingshot successfully connects to master meta-controller, slingshot instructs it to take control over the remaining compute nodes—the master establishes an secure connection to each of those compute nodes and instructs them to act as slaves.

After that, slingshot sends commands to the master meta-controller, which properly distributes the commands to the slave meta-controllers, if needed. Each command specifies the target compute node or one of its containers on the compute node, where the command is expected to run. A command can be either a blocking command or a nonblocking command. If it is a blocking command, the meta-controller will wait until the the command finishes execution and sends back the result of the command (i.e., the exit status) to slingshot. If the command is a nonblocking command, the meta-controller forks a separate process to handle the command and immediately responds to slingshot. Our current implementation uses blocking commands to set up the containers and the emulation infrastructure, and uses nonblocking commands to start the simulator and the user applications within the containers.

To set up the experiments correctly, the meta-controllers on the compute nodes need to run a series of commands:

1) *Set up MPI.* The master meta-controller creates the machine file and instructs the slave meta-controllers to generate the necessary keys for MPI to enable SSH logins without using passwords.
2) *Create containers.* The meta-controller creates a container for each collocated emulated host on the compute node. This step also includes creating the union file systems for the containers.
3) *Set up the emulation infrastructure.* For collocated emulated hosts, this step includes installing the virtual Ethernet devices in the containers, creating and configuring the software bridges and TAP devices, and then connecting the network devices to the bridges. For remote emulated hosts, this step includes setting up the OpenVPN server(s).
4) *Run the experiment.* The partitioned network model is distributed among the compute nodes. The master meta-controller initiates the MPI run, which starts the simulator instances on each compute node with the partitioned model.
5) *Start applications within containers.* Individual commands are sent to the meta-controllers to install and run applications at the emulated hosts.
6) *Shut down the experiment.* At any time, one can shut down the experiment by terminating the simulator, stopping the containers, and removing the emulation infrastructure (such as the bridges and the TAP devices).

All these commands are issued from slingshot automatically using the meta-controller command framework.

IV. EXPERIMENTS

In this section we describe the set of experiments we performed to validate the accuracy of our testbed, and determine its performance limitations, in order to show the utility of our approach. The experiments described in this section are conducted on a prototype PrimoGENI cluster with eight Dell PowerEdge R210 rack-mount servers, each with dual quad-core Xeon 2.8 GHz processors and 8 GB memory. The servers are connected using a gigabit switch.

A. Validation Studies

We validate the accuracy of the test by comparing the TCP performance between emulation and simulation. We use TCP for validation because it is highly sensitive to the delay jitters

and losses and therefore can magnify the errors introduced by the emulation infrastructure. Previously we performed validation studies for the VPN-based emulation infrastructure, which is designed for connecting applications running on remote machines [21]. In this study, we focus only on the emulation infrastructure based on software bridges and TAP devices, which we use to connect the collocated emulated hosts with the simulator instances run on multiple compute nodes.

We first compare the TCP congestion window trajectories achieved by the real Linux TCP implementations on the OpenVZ containers against those from our simulation. We arbitrarily choose three congestion control algorithms—BIC, HIGHSPEED, and RENO—out of the 14 TCP variants we have implemented in the PRIME simulator [23]. We use a dumbbell model for the experiments. The dumbbell model has two routers connected with a bottleneck link with 10 Mb/s bandwidth and 64 ms delay. We attach two hosts to the routers on either side using a link with 1 Gb/s bandwidth and negligible delay. The buffers in all network interfaces are set to be 64 KB. In the experiment, we direct a TCP flow from one host to the other that traverses the two routers and the bottleneck link. For each TCP algorithm we test three scenarios. In the first scenario, we perform pure simulation and use a simulated traffic generator. In the second and third scenario, we designate the two hosts as emulated hosts and use iperf to generate the TCP flow (and measure its performance). We run the emulation on one compute node for the second scenario, in which case the compute node runs simulator in container 0 and two other containers as the emulated hosts. In the third scenario, we use two compute nodes, each running one simulator instance and one emulated host. The emulated traffic in this case has to travel across the memory boundary between the two compute nodes in simulation (using MPI). For simulation, we use a script to analyze the trace output; for emulation, we sample the (/proc/net/tcp) file at regular intervals to extract the TCP congestion window size. Fig. 4 shows very similar TCP congestion window trajectories between simulation and emulation.

Next, we use a TCP fairness test to show whether our approach can correctly intermingle emulated and simulated packets. We use a similar dumbbell model; however, in this time, we attach two hosts on either side of the routers. We generate two TCP flows in the same direction—one for each of the two hosts on the left side to one of the two hosts on the right side. We select the TCP HIGHSPEED algorithm for both flows. We start one flow 20 seconds after the other flow. We compare two scenarios: in the first scenario we perform pure simulation; and in the second scenario, we designate the two end hosts of the first TCP flow as emulated hosts. The results are shown in Fig. 5. For both scenarios, we see that the congestion window size of the first TCP flow reduces when the second TCP flow starts; both flows eventually converge with a fair share of the bandwidth (at about 30 seconds after the second flow starts transmitting). Again, we see very similar results between simulation and emulation.

The emulation infrastructure inevitably puts a limit on the

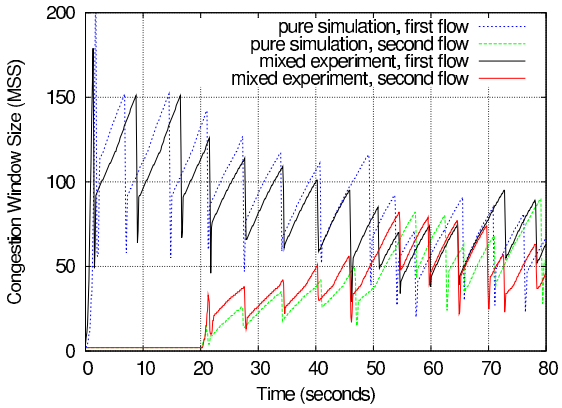


Fig. 5. Similar TCP fairness behavior between simulation and emulation

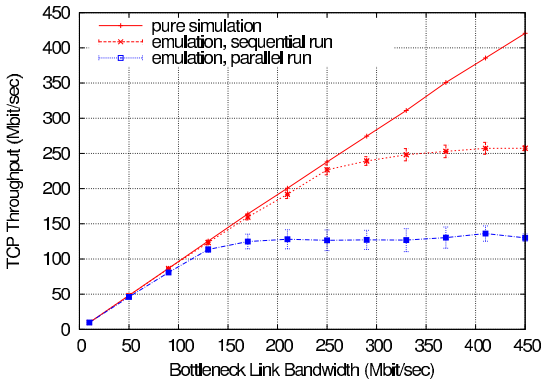


Fig. 6. TCP throughput limited by emulation infrastructure

throughput of the emulated traffic. In the last validation experiment, we look into this limiting effect on the emulated TCP behavior. Again, we use the dumbbell model. To increase the TCP throughput, we reduce the delay of the bottleneck link of the dumbbell model to 1 millisecond. For the experiment, we vary the bandwidth of the bottleneck link from 10 Mb/s to 450 Mb/s with increments of 40 Mb/s. Like in the first experiment, we direct a TCP flow from one host to the other through the bottleneck link and we compare three scenarios: the first using pure simulation, the second with an emulated flow within one compute node, and the third with an emulated flow across two compute nodes. Fig. 6 shows the results. While the throughput increases almost linearly with the increased bandwidth for the simulated flow, the error becomes apparent for emulated traffic at high traffic intensity. The throughput for the emulated traffic is kept below roughly 250 Mb/s for sequential runs and 130 Mb/s for parallel runs. The reduced throughput for parallel runs is due to the communication overhead as the TCP traffic gets exposed to the additional delay between the parallel simulator instances. Since emulation accuracy heavily depends on the capacity of the emulation infrastructure. We look into the emulation performance in more detail in the next section.

B. *Performance Studies*

In the previous experiment we see that the emulation infrastructure is placing an upper limit on the emulation traffic the system can support. Here we use a set of experiments

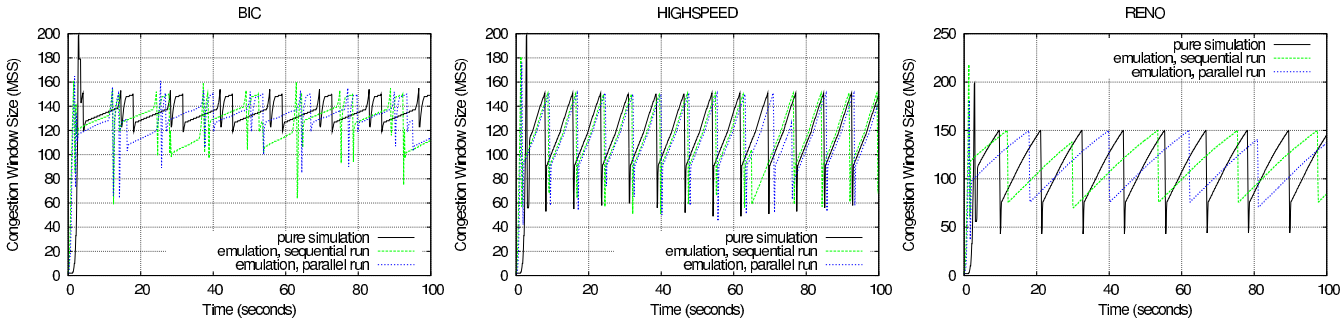


Fig. 4. Similar TCP congestion window trajectories between simulation and emulation

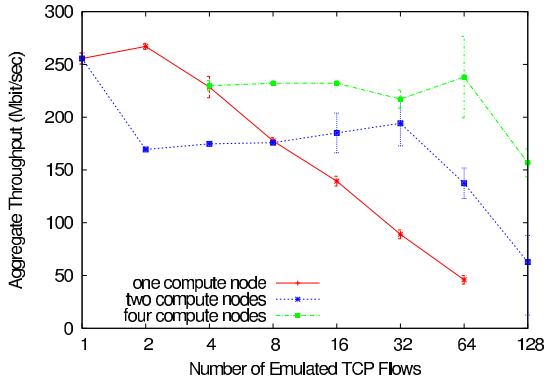


Fig. 7. Aggregate TCP throughput vs. emulated TCP flow count

to measure the capacity of the emulation infrastructure. We use the same dumbbell model (with a bottleneck link of 1 Gb/s bandwidth and 1 millisecond delay), and attach the same number of emulated hosts on each side of the dumbbell routers. We start a TCP flow (using iperf) for each pair of emulated hosts, one from each side of the dumbbell. So the total number of TCP flows is half the number of emulated hosts. We make same number of the TCP flows to go from left to right as those from right to left. We vary the number of emulated hosts in each experiment. We show the sum of the measured throughput (from iperf) for all emulated TCP flows in Fig. 7.

In the first experiment, we assign all emulated hosts to run on the same compute node. For one flow, the throughput reaches about 250 Mb/s, as we have observed in the previous experiment for sequential runs. The aggregate throughput increases slightly for two flows, but drops continuously as we increase the number of flows all the way to 64 flows (that's 128 VMs). The slight increase is probably due to TCP's opportunistic behavior that allows it achieve better channel utilization with more flows. We suspect that the drop in throughput is because the emulated hosts (OpenVZ containers) are competing for shared buffer space in the kernel network stack. With a smaller share when the number of containers gets bigger, the TCP performance degrades.

In the second experiment, we divide the network between two compute nodes (splitting the model along the bottleneck link). The throughput for one emulated flow in this case is around 130 Mb/s, like we have observed before. As we increase the number of flows, the aggregate throughput increases slightly until 32 flows, after which we start to see a significant drop. Running the simulation on two compute nodes results in more the event processing power, the simulator is thus capable of handling more emulated flows than the sequential case.

In the third experiment, we extend the dumbbell model by placing four core routers in a ring and connecting them using the bottleneck links. We attach the same number of emulated hosts to each core router. Each TCP flow was sent from an emulated host attached to one router to another emulated host attached to the next router in the ring. In this case, we spread the number of emulated flows evenly among the emulated hosts. Comparing with the scenarios with two compute nodes, the aggregate throughput for the four-node case is higher. Again we think is is due to the higher processing power of the parallel simulator. The throughput starts to drop for 128 flows (that's 32 VMs per compute node) as they compete for the shared buffer space.

V. SUMMARY AND FUTURE WORK

PrimoGENI supports real-time network simulation and emulation on GENI enabled resources. It provides the necessary tools for the experimenters to construct and configure the network models, allocate the resources, deploy and run the experiments, and collect the experiment results. Each experiment is run within its own slice of resources, allocated using the GENI control framework. The resources may consist of the compute nodes in the cluster and possibly remote machines. Each compute node is treated as a scaling unit in PrimoGENI; it can be configured to run a parallel simulator instance together with a set of virtual machines for emulated hosts. The latter provides a realistic operating environment for testing real implementations of network applications and services. PrimoGENI applies parallel and distributed simulation techniques to synchronize the simulator instances running within the scaling units. The virtual machines are connected with the simulator instances using a flexible emulation infrastructure.

Our validation experiments show that PrimoGENI can produce accurate emulation results when the emulated traffic does not exceed the capacity of the emulation infrastructure. Our performance studies show that the throughput of the emulated traffic is dependent upon the number of virtual machines running on each scaling unit.

We are currently investigating more efficient mechanisms to increase the emulation throughput. For example, we want

to apply zero-copy techniques for moving data packets more efficiently between the virtual machines. We also want to have a fine-grained control of the I/O functions inside the real-time simulator to reduce the overhead. Another technique is to slow down the emulation speed using time dilation [33], which controls a systems notion of how time progresses on the virtual machines. This is achieved by enlarging the interval between timer interrupts by what is called a time dilation factor (TDF), thus slowing down the system clock. If we proportionally slow down the real-time simulator, the applications can experience a higher networking speed than what can be offered by the emulation system.

We will extend PrimoGENI to allow the virtual network in a slice to be able to connect to other GENI resources to facilitate larger network experiments. We will also implement the mechanism to control, inspect, and visualize the state of the network experiment. To access and modify the runtime state, we need to implement a dynamic query architecture based on the meta-controller design. Our design will consist of three components: a database that runs separately from the simulator, the data managers for collecting the states in simulation, and the query gateways which ferry query requests and results between the database and the data mangers. We will investigate strategies for complex queries involving collectieve operations on the large network state.

ACKNOWLEDGMENTS

We would like to thank our undergraduate students, Eduardo Tibau and Eduardo Peña, for the development of the slingshot IDE. The PrimoGENI project is part of GENI's Spiral 2 prototyping and development effort, funded by the National Science Foundation (NSF) through GENI Project Office (GPO). The PrimoGENI project is built on PRIME, which is also funded by NSF through the grant CNS-0836408.

REFERENCES

[1] GENI, http://groups.geni.net/.
[2] GENI System Overview, http://groups.geni.net/geni/wiki/GeniSysOvrvw.
[3] PRIME, http://www.primessf.net/.
[4] J. Cowie, D. Nicol, and A. Ogielski, "Modeling the global Internet," *Computing in Science and Engineering*, vol. 1, no. 1, pp. 42–50, 1999.
[5] P. Barford and L. Landweber, "Bench-style network research in an Internet instance laboratory," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 21–26, 2003.
[6] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the Internet," in *Proceedings of the 1st Workshop on Hot Topics in Networking (HotNets-I)*, 2002.
[7] N. Spring, L. Peterson, A. Bavier, and V. Pai, "Using PlanetLab for network research: myths, realities, and best practices," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, pp. 17–24, 2006.
[8] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker, "Scalability and accuracy in a large scale network emulator," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, 2002, pp. 271–284.
[9] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, 2002, pp. 255–270.
[10] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford, "In VINI veritas: realistic and controlled network experimentation," in *Proceedings of the 2006 ACM SIGCOMM Conference*, 2006, pp. 3–14.
[11] J. DeHart, F. Kuhns, J. Parwatikar, J. Turner, C. Wiseman, and K. Wong, "The open network laboratory," *ACM SIGCSE Bulletin*, vol. 38, no. 1, pp. 107–111, 2006.
[12] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh, "Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols," in *Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2005)*, 2005.
[13] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu, "Advances in network simulation," *IEEE Computer*, vol. 33, no. 5, pp. 59–67, 2000.
[14] G. F. Riley, "The Georgia Tech network simulator," in *Proceedings of the ACM SIGCOMM Workshop on Models, Methods and Tools for Reproducible Network Research (MoMeTools'03)*, August 2003, pp. 5–12.
[15] G. Yaun, D. Bauer, H. Bhutada, C. Carothers, M. Yuksel, and S. Kalyanaraman, "Large-scale network simulation techniques: examples of TCP and OSPF models," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 3, pp. 27–41, 2003.
[16] J. Liu, "A primer for real-time simulation of large-scale networks," in *Proceedings of the 41st Annual Simulation Symposium (ANSS'08)*, 2008, pp. 85–94.
[17] K. Fall, "Network emulation in the Vint/NS simulator," in *Proceedings of the 4th IEEE Symposium on Computers and Communications (ISCC'99)*, 1999, pp. 244–250.
[18] R. Simmonds and B. W. Unger, "Towards scalable network emulation," *Computer Communications*, vol. 26, no. 3, pp. 264–277, 2003.
[19] X. Liu, H. Xia, and A. A. Chien, "Network emulation tools for modeling grid behavior," in *Proceedings of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid'03)*, 2003.
[20] J. Zhou, Z. Ji, M. Takai, and R. Bagrodia, "MAYA: integrating hybrid network modeling to the physical world," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 14, no. 2, pp. 149–169, 2004.
[21] J. Liu, Y. Li, N. V. Vorst, S. Mann, and K. Hellman, *J. Systems & Software*, vol. 82, no. 3, pp. 473–485, 2009.
[22] J. Liu, Y. Li, and Y. He, "A large-scale real-time network simulation study using PRIME," in *Proceedings of the 2009 Winter Simulation Conference*, December 2009.
[23] M. Erazo, Y. Li, and J. Liu, "SVEET! A scalable virtualized evaluation environment for TCP," in *Proceedings of the 5th International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom'09)*, April 2009.
[24] J. Liu and Y. Li, "On the performance of a hybrid network traffic model," *Simulation Modelling Practice and Theory*, vol. 16, no. 6, pp. 656–669, 2008.
[25] ProtoGENI, http://www.protogeni.net/.
[26] ORCA, https://geni-orca.renci.org/trac/.
[27] METIS, http://www.cs.umn.edu/~metis/.
[28] OpenVZ, http://wiki.openvz.org.
[29] FunionFS, http://funionfs.apiou.org/.
[30] ebtables, http://ebtables.sourceforge.net/.
[31] OpenVZ Virtual Ethernet Device, http://wiki.openvz.org/Veth.
[32] Apache MINA, http://mina.apache.org/.
[33] D. Gupta, K. Yocum, M. McNett, A. Snoeren, A. Vahdat, and G. Voelker, "To infinity and beyond: time-warped network emulation," in *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI'06)*, 2006.

A Pragmatic Approach to Wireless Channel Simulation in Built Environments

Ryan McNally, Matthew Barnes and DK Arvind
Centre for Speckled Computing
School of Informatics, University of Edinburgh
Edinburgh EH89AB, Scotland, UK
{mbarnes, dka}@inf.ed.ac.uk

***Abstract*—The performance estimation of Wireless Sensor Networks (WSNs) is recommended before large scale deployment as the cost of redesign after the event could be expensive. This paper presents a novel wireless channel model, which exposes WSN applications such as environmental data gathering to realistic conditions, such as non-isotropic radiation, unreliable and asymmetric links, and the effect of building structures in the deployment area. Applications that can cope with these phenomena in simulations are more likely to run to specification on the real network.**

I. Introduction

There is often a mismatch between the performance predicted using simulations and the actual performance delivered in a deployed Wireless Sensor Network (WSN). This is often the case in built environments thanks to the vagaries of communication due to obstructions from walls and furniture, and people moving around the space. Therefore any improvement in accurately predicting battery life-times of the WSN nodes will be a step towards their successful deployment. For example, a common solution to extend the operational lifetime of WSN nodes is to reduce their duty cycle, but this may violate temporal requirements such as the resolution of the sensed data.

The network topology also informs the longevity of the wireless sensor network. Bottlenecks in the network graph often leads to concentrations of network traffic, which exerts an additional energy burden on particular nodes leading to their early demise, and which in turn results in certain sections of the network being unreachable. There is a risk that the deployment would not meet the specification and resolving the issue after deployment could be quite expensive. The approach in this paper aims to formulate models which reflect the reality of deployment, i.e. not too optimistic that the capability of the network is overestimated, and yet not too pessimistic that the nodes are over specified and the resulting network is overly expensive. In this paper we will examine some of the difficulties in modelling a sensor network deployment, and in particular modelling the wireless channel that determines how the simulated nodes communicate.

The channel model presented in this paper does not attempt to model in detail the physics of radio signal propagation; indeed such a model would be prohibitively complex and computationally expensive. The proposed model instead aims to capture certain classes of errors and difficulties that real networks encounter, and which are not considered by current models. These include issues such as non-isotropic performance and the effects of obstructions in the deployment area. Errors such as intermittent communication and asymmetric network links are common in deployed sensor networks, and it is important that they are considered when analysing the network behaviour. The proposed channel model can be easily enhanced with experimental data captured from a prospective deployment site, and includes a mechanism to model building features and their effects on the wireless channel. In the rest of this paper, Section II outlines existing channel model simulation and related work. Section III describes proposed channel model. Section IV describes the simulation environment. Section V presents a case study using the model. Section VI outlines planned future work. Section VII presents the conclusion of the paper.

II. Related Work

The channel model in a simulator can vary in complexity from a simple range threshold check to a full simulation of the radio signals interaction with the environment.

A naive model, termed as the unit disk graph model, assumes that radios have an effective range of transmission. Other radios within this range will receive every transmission, whereas those outside will receive none. This model is often used in the early stages of development of algorithms for target WSNs, such as debugging their behaviour. However, networks using such a model will feature fully-reliable and bidirectional links, and algorithms developed on this basis are unlikely to operate successfully in a real deployment.

The unit disk model typically represents signal interference in a similarly simplistic manner: when a radio is deemed to be within the area of effect of two concurrent transmissions, then it receives neither. This disregards the fact that radio signals still contribute to interferences even if they are not receivable. An extension to the model to address this shortcoming is to define a further interference range, which is larger than the reception range, and in which a transmission will interfere with an in-progress reception.

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

In practice most simulators model the path loss, i.e. the dissipation of radio signals in the environment due to factors such as distance, reflection, diffraction and scattering. The energy measured by a receiver is the transmitted energy, less the path loss over the distance, and should the received energy rise above the noise floor of the radio then the packet is received successfully. Two terms determine the models behaviour: a path loss exponent, i.e. the rate at which a signal decays in the environment, and a random term with a parameterised normal deviation that controls the variation of the received power from the idealised inverse-square relationship with distance. These two terms, usually denoted n and σ, control the transitional region of radio coverage [1]. This is the range of distance over which the high Packet Reception Ratio (PRR) experienced when close to the transmitter transitions to the low PRR experienced further away.

Molina-Garcia-Pardo *et al.* [2] describe how these values were determined by deploying a spectrum analyser at set distances from a sensor node deployed in different environments. Their results showed that the values vary significantly for the different environments, and that the values were different for short and long ranges.

Zhou *et al.* [3] improved the model of radio energy emissions by including non-isotropic radiation. This model is similar to the interference-extended unit disk model described above, in that there is a reception range and an interference range that control how nodes communicate. The difference however is that these ranges also depend on the angle between transmitter and the receiver. The regions of communication and reception are no longer circles, but irregular shapes controlled by a parameter in the model. The results presented with this model demonstrated the importance of considering the unreliability of radios when analysing networked algorithms, as it has a noticeable impact on the efficacy of MAC protocols and routing algorithms. Any applications relying on these services will be affected proportionately.

Although the radiation from an antenna is not isotropic, neither is it entirely random and irregular. Modelling the energy emission pattern of a radio as being entirely random will with aggregation over sufficient number of simulations converge towards isotropic results. This will disregard the consistent performances that real radios do exhibit for a given transmission direction. The non-random, non-isotropic nature of antenna patterns has been addressed by so-called sectorisation techniques, where the area around the transmitter is divided into sectors, and the path-loss terms are measured in individual sectors.

Robinson *et al.* [4] used such a scheme to reduce the number of measurements needed to characterise the coverage of an 802.11 network deployed in an urban environment. Their approach involved creating a rough map of different terrain types according to the nature of the buildings in different areas. This map was then used to estimate sector directions and widths around each WiFi mesh node that have roughly monotonic terrain types, taking care to reduce the number of different sectors and therefore the measurements required on the ground. For each sector, a number of measurements were made to determine the distance within which acceptable performance is obtained. A coverage map of the node was computed and areas of poor performance were identified, so that additional mesh nodes could be targeted to provide full coverage. Whereas such an approach reduces the workload for characterising a small outdoor network, it scales poorly for large indoor sensor network with hundreds or even thousands of nodes, and will be disruptive to users of the building.

Chipara *et al.* [5] have proposed a path-loss model which accounts for the walls in the building. The walls are classified according to their material and effect on transmissions. When simulating a transmission, the impact of the intervening walls between the transmitter and the receiver are accounted for in the power calculation. Experimental results are presented that demonstrate the importance of considering walls and other obstructions in the channel model. A tool helps in the classification of the walls in the deployment area based on transmissions from a small number of points in the network. The channel model assumes isotropic radiation from nodes, and the results presented show that the effects of walls outweigh that of directional radiation, and the burden of the additional measurements is not justified. The model considers the layout and material of the walls in the deployment area, but are simplified to be geometric planes with zero width. The effect of wall thickness may be significant with certain node layouts.

III. Channel Model

The rationale for a new model was to address some of the limitations of path-loss models for networks deployed in built environments by accounting for the directional effects of antenna radiation and the resulting impact on deployment strategies. Current path-loss models also assume that the channel model characteristics are homogeneous over the entire deployment area which in reality is unlikely as subtle details of the environment and deployment would impact the radio channel. The proposed model is described in terms of the expected probability of a given packet being received, termed as the Packet Reception Rate (PRR). This value varies with distance and angle from the transmitter and can be easily measured by the sensor nodes themselves, requiring no special equipment such as signal analysers. A PRR map is constructed around each transmitter and the receivers location determines the probability of receiving a packet as read from the PRR map.

A. Construction of the PRR map

The PRR map was constructed using SunSpot [6] devices equipped with a CC2420 radio and an F-type antenna, by taking measurements on a 2-D grid around the transmitter-receiver pair for a set number of packet transmissions, and the PRR measure was calculated as the fraction of packets which arrived at each point in the grid. The radio output power was set to the minimum possible value in order to keep the number of points in the grid down to a manageable level. In

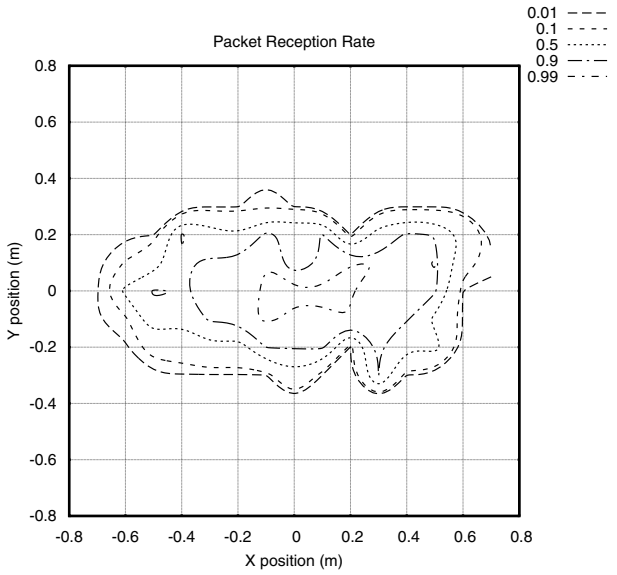


Fig. 1. PRR map of SunSpot device

the case of the PRR map shown in Figure 1, the transmitter was fixed on a wooden surface with the receiver moved around in a grid pattern at intervals of $10\ \mathrm{cm}$. At each grid point the central node was set to transmit one hundred radio packets at intervals of $50\ \mathrm{ms}$ between each packet. The receiver relayed back the number of packets received to a base-station where the data was logged. The PRR pattern was roughly ellipticall with a number of small, but albeit significant, irregularities. The second graph shows how PRR varies with distance from the transmitter on one axis. These measurements were repeated four times over a period of two days. Although small variations were observed between each PRR map, their essential features were largely preserved.

B. Synthesis of the PRR map

The basis of the synthetic PRR map is an ellipsoid, defined by three axial radii a, b and c. To determine the PRR pattern between a transmitter and a receiver, the receiver's position is first transformed into the transmitter's frame of reference, and the 3-D spherical coordinates (θ, φ, r_p) of this point are computed. The radius of the bounding ellipsoid in this direction is calculated as

$$r_e = \sqrt{\frac{a^2}{\cos^2\theta \sin^2\varphi} + \frac{b^2}{\sin^2\theta \sin^2\varphi} + \frac{c^2}{\cos^2\theta}}. \qquad (1)$$

The ratio, d_q, of r_p and r_e, gives the distance between the transmitter-receiver pair in terms of the radius of the bounding ellipsoid. If the PRR was assumed to decrease linearly with distance, then one could use this directly to determine the reception of a packet or otherwise. However PRR does not scale down linearly with distance as can be observed in Figure 1.

This issue is addressed in the model by employing a user-defined, point-based function to approximate the variation of PRR with the ellipsoidal radius. This function is used to determine the PRR for a given radial distance quotient. Although this model exhibits the elliptical, non-linear fall-off in PRR as observed in Figure 1, it still lacks the smaller irregularities. These features are simulated by a number of randomly-generated modulations to the ellipsoid, termed as lobes, that alter the distance quotient d_q. The lobes are of two types:

To simulate these irregular features we introduce a number of randomly-generated additions to the ellipsoid, termed lobes, that alter the distance quotient d_q. The lobes are of two types:

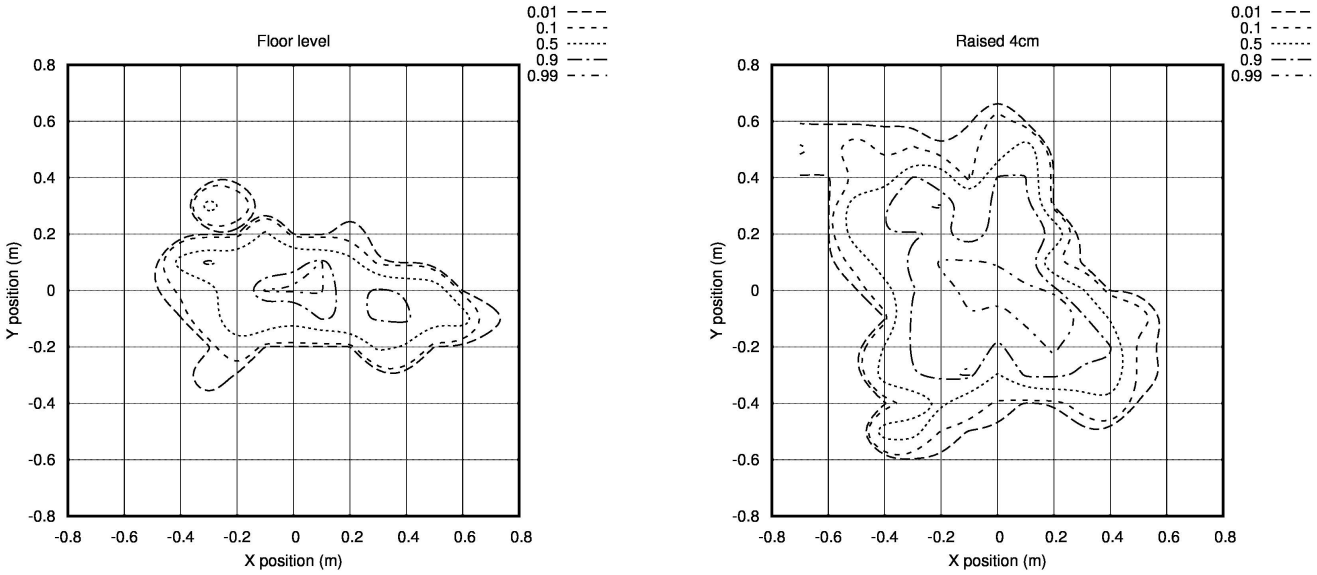
- Point-based lobes alter d_q based on the receivers distance from a randomly-generated point.
- Ray-based lobes alter d_q based on the angle between a randomly-generated vector and the vector from the transmitter to the receiver.

The influences of both varieties of lobes are determined randomly within user-defined ranges, and lobes may make either a positive or negative contribution to d_q. An example of a synthetic PRR map generated with this addition is depicted in Figure 2. In comparison with Figures 1 and 2, the features of the PRR maps derived experimentally are reproduced mainly elliptical, a non-linear fall in PRR with distance together with small-scale irregularities. One should not expect exact correspondences between the real and synthetic PRR maps, any more than the variations observed between maps measured in the same environments on different days.

Another source of substantial variation is the nature of the surface on which the devices are placed which will alter the channel and affect the performance of the radio as illustrated in Figure 3. The first image shows the PRR map of a SunSPOT device where both the transmitter and the receiver were lying directly on a wooden floor. The second PRR map was measured in the same room with the same devices placed on top of $4\ \mathrm{cm}$ tall plastic boxes. Any model that reproduces faithfully variations due to the physics of interaction of the radio with the environment would be quite complex and labourious to build and defeats the purpose of saving time and effort by performing network simulations.

IV. The Channel Environment

The synthetic PRR map is designed to expose WSN applications to network problems normally encountered in real network deployments, but it does not model the impact of the surrounding environment. The nature of this environment would obviously affect radio propagation, e.g., indoor plants will absorb transmissions to varying degrees according to its density and water content. The $2.4\ \mathrm{GHz}$ operating frequency and low transmission power of sensor networks also mean that a reliable link is unlikely to be established if two nodes are blocked by structures. The impact of features in the



(a) Floor Height Measurement

(b) Raised Measurement

Fig. 3. PRR maps of similar deployments

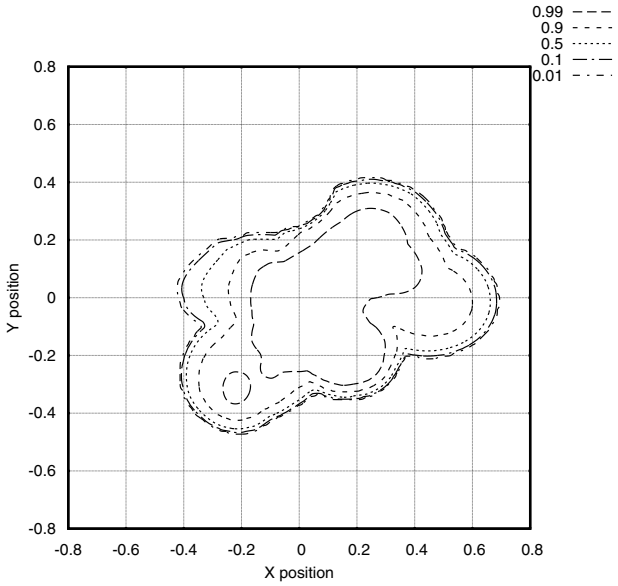


Fig. 2. Synthetic PRR map

deployment area and the layout of the built environment must be considered during simulation.

The channel model addresses the large-scale characteristics of the radio channel by using an image-based map of the deployment area to inform the model by representing obstructions. An image of the floor-plan is converted to greyscale to give a continuum of values to represent different levels of transmission obstruction. A point-based function is used to map the grey levels to obstruction strengths. A histogram of pixel intensity values present in the map image is displayed to the user to aid in setting appropriate levels.

The map image and the weight function are used to alter the effective distance between the transmitter and receiver. When the receiver's spherical coordinates are computed, the line segment that lies between the transmitter and receiver is laid over the map image and the intersecting pixels examined. The effective distance between the two nodes is calculated as

$$d_e = \sum d_i \times w_i, \tag{2}$$

where, d_i is the length of the intersection of the line segment and the pixel, and w_i is the obstruction weight of that pixel. This effective distance, d_e, is used to calculate the PRR as before. Pixels with obstruction weights of greater than one will inhibit transmissions by increasing the effective distance and so reducing the PRR.

The environment does not always affect transmissions in a deleterious manner. Multi-path interference, where reflections of transmissions combine with the signal that travelled directly from the transmitter to the receiver, can be constructive as well as destructive. In particular, experiments conducted with the SunSpot devices have shown that transmissions along the axis of a corridor have a greater effective range than those in an open space thanks to constructive interferences.

The map-image based channel model can also represent this phenomenon by assigning a weight less than one to pixels in a corridor. This has the effect of increasing the effective range of a transmission that passes through these pixels.

Note that this would also slightly increase the effective range of transmissions that travel perpendicularly to the corridor's axis that is unlikely to benefit from the constructive multi-path reflections. However, the increase in range would be minor as fewer corridor pixels are encountered in the transection, and the penalty of passing through the corridor's two bounding walls would likely overwhelm any strength accrued.

Map images have proven to be a convenient way to express spatial variations in the PRR map-based channel model, which could be equally applied to models with a different basis.

A. Buildings

For WSNs deployed within buildings, features such as corridors can have unexpected effects, plasterboard walls will impede transmissions to a lesser degree than concrete, and therefore the layout and material of internal walls is of interest. The size and composition of furniture within the building impact on the channel.

Buildings are modelled in the channel model using floor-plan images. Pixel intensities are combined with a weight function to influence transmissions that pass over them. Buildings also raise the possibility of deployment in three dimensions, and so multiple images are used per building, one for each floor. Each level of a building is assigned a thickness and a weighting value that affects transmissions that travel through the floor material. Modern buildings are often designed with areas without a floor such as atria. These features are represented by transparent pixels in the floor-plan image, and so transmissions between floors pass through them unimpeded.

B. Construction of building models

The use of floor-plan images and functions that map between pixel intensity and channel obstruction weight provide a convenient mechanism for simulating network performance in indoor wireless networks. This should be supplemented with appropriate data that reflect reality: accurate floor-plan image of the building, and the weight values for different wall types. Floor-plan images are typically obtained from architectural schematics detailing the position and composition of walls. Even if the schematics lack exact specifications of wall material, it is unusual for a large number of different types to be used in the same building. This simplifies the process of identifying wall types in the floor-plan image and designating them with an appropriate pixel intensity.

Assigning obstruction weights is a slightly more involved process, requiring experimentation with the WSN device. A small number of wall material types in a typical building also helps in this case by limiting the number of measurements required. Measuring the PRR between devices by varying inter-device distance, both with and without the presence of a representative wall, enables the contribution of the latter to the channel model to be estimated and a weight assigned. For a network already in place, an automated wall classification technique such as the scheme detailed in [5] may be employed. As noted previously, the model that this technique is designed to work with does not consider the overall thickness of wall material that signals must pass through, and so such a scheme would have to be altered to take this into account.

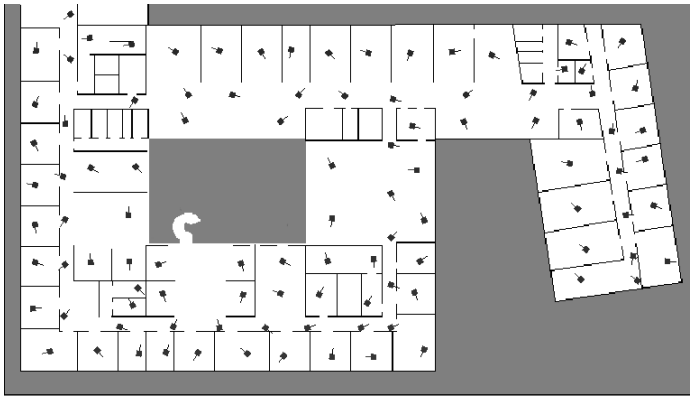


Fig. 4. Node deployment

The ease of measuring and incorporating the PRR data in the channel model means that it can be refined continually after deployment. WSN devices in the network monitor the PRR of their links, and relay this information back to the base-station as they would any other sensed data. These readings can then be compared against simulated results and any discrepancies investigated. A number of issues can affect network link quality that cannot be inferred from architectural drawings, and which cannot be predicted in advance without an exhaustive assessment of the deployment area. These factors include interference from other wireless signals, additional communications obstructions caused by furniture, or even the density of people moving in a given area.

Altering the weight function allows deployment-wide discrepancies to be corrected, while the deployment map image allows localised alterations to be made to the channel model. Any non-trivial deployment is likely to have many of these special cases and areas of divergent channel behaviour, and so the ability to address them on a case-by-case basis is a powerful tool in refining the channel models.

V. The Channel Environment

A case study based on a single floor (see Figure 4) of the Informatics Forum building at the University of Edinburgh illustrates the importance of accounting for radio emission patterns and building layout when simulating wireless sensor networks. White areas represent the floor, black lines represent walls, while the grey areas are voids and the WSN nodes are placed in this space with random orientations.

A typical sensor network application gathers data from all the nodes to one or more sinks, with the data being routed to the sinks via multi-hop paths. The nature of the network connectivity graph will have a significant impact on

the performance of the data routing algorithms. It is therefore essential that the simulation of such a network deployed in a building includes the effects of the building environment on PRR and network connectivity. A few experiments were conducted in the building to establish the parameters of the building environment that affect the PRR and network connectivity.

A sensor network, communicating in the $2.4\,\mathrm{GHz}$ ISM wireless band was deployed with a single device in each office and along the corridors spaced at approximately five metre intervals. All the devices were programmed to transmit a data packet every second, using a CSMA Medium Access Control (MAC) protocol, and to record the number of packets received from each source device encountered. Devices also recorded the number of successful transmissions they had made each minute as the MAC protocol does not guarantee that a transmission can be made.

Every minute each device transmitted the number of transmissions it made, the sources recorded and packet counts in the last minute to an additional node connected to a PC where the statistics were recorded. The statistics transmissions were routed over multiple hops and were not counted in the recorded counts. Nodes were placed in positions in each room suitable for long term deployment and with no specific orientation. The experiment was left to run for twenty-four hours, at which time the orientation of each node was rotated by 90 degrees.

The average PRR for each link was calculated based on the recorded transmission and reception counts. Two categories of links were created to simplify the illustration of these results in Figure 5: *good links* with PRR greater than 75%, and *poor links* with PRR values between 25% to 75%, and links with PRR value lower than 25% were discarded.

The following general rules were established:

- Transmissions can penetrate one inter-office wall, but not two.
- Transmissions performed well with line-of-sight down corridors or across the atrium.

As suggested by the PRR map in Figure 1, device orientation has a significant impact on the link range, leading to consistent unidirectional links between a pair of nodes with given orientations. These characteristics must be respected in a connectivity graph in order for simulation results generated to have any credence.

The connectivity graph of a standard path-loss channel model, two examples of which are depicted in Figure 6, displays realistic network phenomena, such as unidirectional and intermittent links. However, a tuning problem is present: altering the channel parameters to satisfy the inter-office link constraints breaks results in too few long-range links between nodes with line-of-sight, while decreasing the path-loss exponent in an effort to produce these long-range links results in radio links that span multiple internal and external walls. For example, the links that span between the two wings of the building, and the high link density in the lower-left of the network. Such links are of particular concern as they offer unrealistic shortcuts for data routing, and would influence any routing performance analyses.

In addition, the unidirectional links are based purely on random factors. Seeding the channel's random number generator with different values, as in the two connectivity maps depicted, will result in completely different sets of such links. This does not reflect the consistent orientation-based performance observed with real devices. This weakness will be apparent in any channel model that does not consider antenna emission patterns and node orientation.

Capturing orientation-based performance in simulation is important as real nodes are likely to be deployed in such a manner as to capitalise on it when possible. For instance, when deploying nodes to form a routing backbone along a corridor, it would be foolish to ignore the additional performance to be gained by correctly orienting the nodes to take advantage of the antenna's energy-emission pattern. Figure 7 shows the connectivity graph from the PRR-map based channel model. Altering the elliptical basis of the PRR map allows device orientation to become significant and produce consistent unidirectional links, while assigning a high impediment weight to wall pixels allows both inter-office and line-of-sight link characteristics to be reproduced. Thus connectivity through walls is limited, while nodes with line-of-sight exhibit stronger performance.

The two connectivity maps shown differ only in the seeds given to the channel's random number generator. It can be observed that the long-range links remain largely consistent in both. The small differences between the two are down to the randomly-generated lobes that are used to introduce irregularity in the PRR maps. This reflects the strong effect of the devices orientation. To evaluate the accuracy of the simulation models, the deployed network shown in Figure 5 was simulated using both the path-loss and PRR map based channel models. The simulations used the floor-plan and node positions of the real deployment area, the antenna orientation was randomised. The resulting connectivity is shown in Table I.

Although the scale of the network is limited, it can be seen that the path-loss based simulation model significantly overestimates the scale and performance of radio connectivity. Whilst the PRR model does not provide a perfect model of the real network, the scale and performance are closer to the measured results. The PRR model provides a more accurate basis for the simulation of higher level activity such as communication protocols and sensor network applications.

VI. Future Work

The current model only considers the static case, and disregards time-variable environmental effects. One example of such an effect in networks deployed outside is the atmospheric moisture content, particularly in the $2.4\,\mathrm{GHz}$ ISM band. Although the atmospheric conditions inside buildings are generally more stable, the presence and movement of people has been shown to have an effect on the performance of 802.11

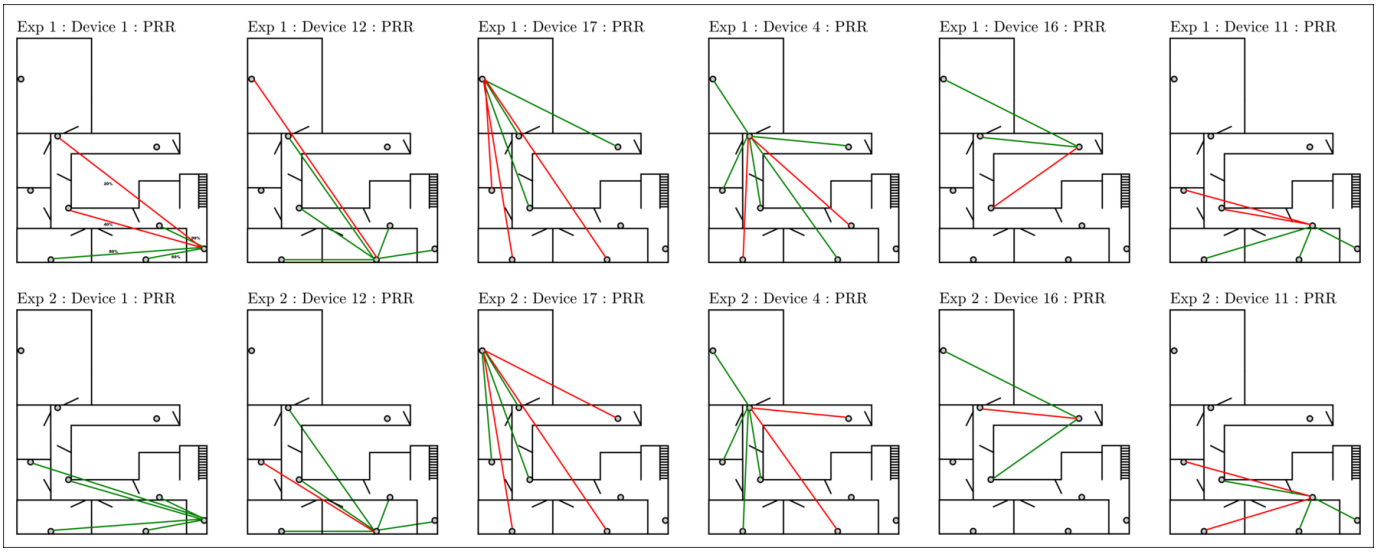
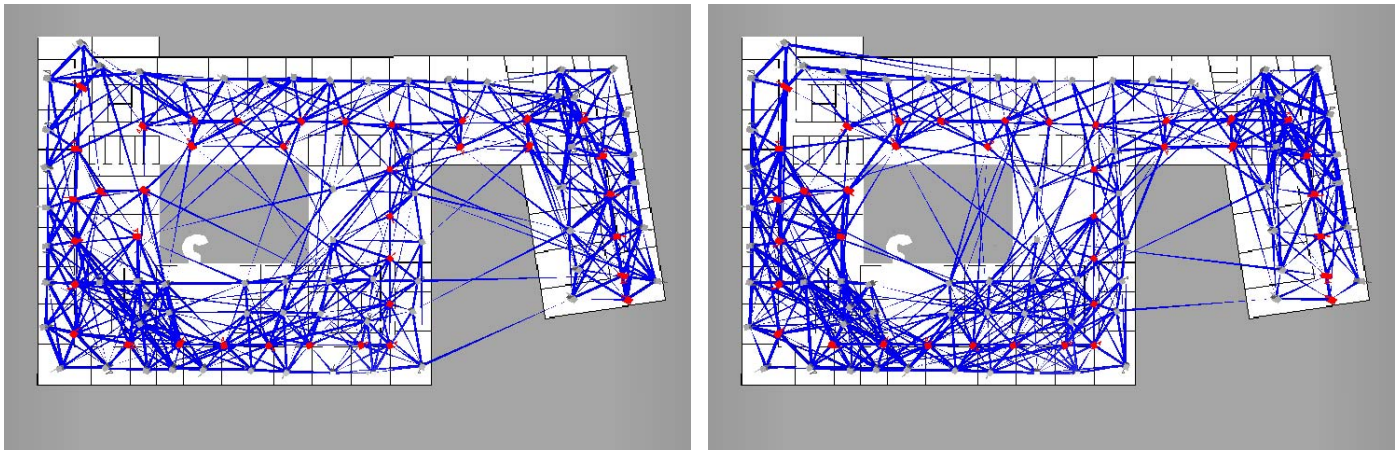
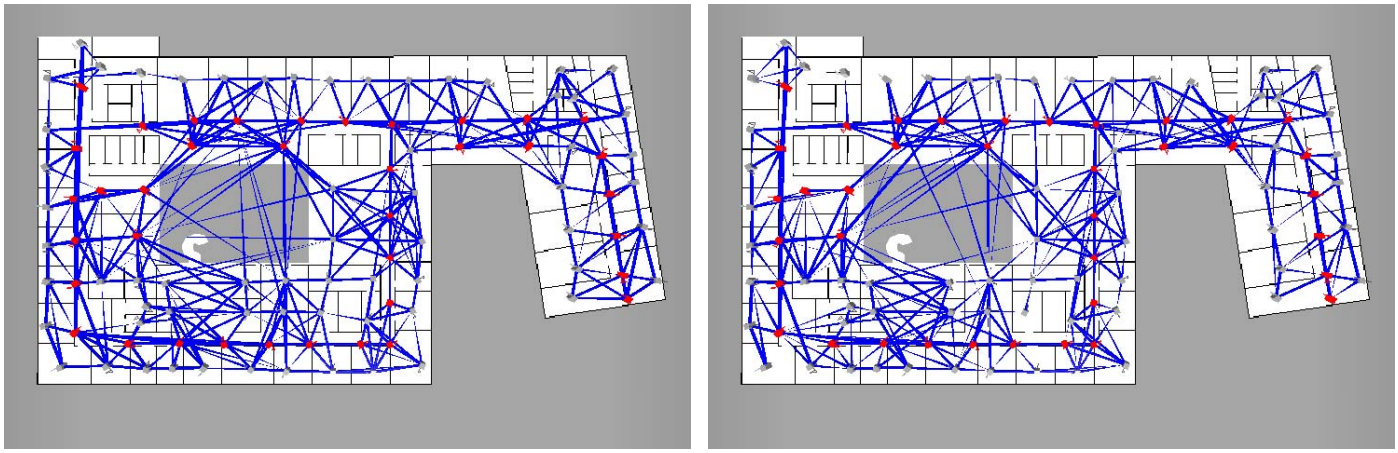


Fig. 5. Device PRR Connectivity



(a) Simulation A (b) Simulation B

Fig. 6. Path-loss connectivity map



(a) Simulation A (b) Simulation B

Fig. 7. PRR-map connectivity map

TABLE I
MODEL CONNECTIVITY COMPARISON

Experiment	Mean Links per node	Mean Links $PRR > 75\%$	Mean Links $25\% < PRR < 75\%$
Real-world	5.5	3.8	1.7
Path Loss model	7.7	6.6	1.1
PRR model	4.9	3.4	1.5

networks [7]. Indeed, this effect has been used as the basis of person-tracking wireless network systems [8].

The introduction of a channel model with time-varying features requires the function that maps from inter-node distance to PRR value and the obstruction map image, can be varied according to reflect the modelled conditions. Once such a mechanism is in place, it must be informed with experimental observations from small-scale test deployments.

The impact of the movement of people is naturally incorporated by altering the impediment map image. Measurements of radio performance can be taken in the presence and absence of occupants, and their contribution computed. This allows static building occupants to be added to the impediment map image. Building occupants naturally move around, and a model of their predicted motion pattern must also be incorporated. Here again, map images demonstrate their usefulness: it is a simple matter to mark out hallways and other likely paths on a floor-plan. This data can then be extracted in the simulation and used to restrict occupants motion to realistic paths.

VII. CONCLUSION

This paper has described a novel channel model which seeks to expose network applications to phenomena that occur in real wireless sensor networks and are often overlooked, such as unreliable and asymmetric links, and non-random, non-isotropic performance. The model uses a map of packet reception rate (PRR) values around each WSN device to determine if a given transmission is received. The PRR map is synthesised based on an ellipsoid, with additional random factors to provide the irregularities encountered with real radios. The synthetic PRR maps are shown to reflect the essential characteristics of the PRR map based on measurements using SunSpot devices.

The channel model also takes account of the characteristics of the network deployment area by using simple image maps. Areas of different channel performance, such as open areas and areas with walls, are represented in the image maps with different pixel intensities, which are then mapped to an obstruction weight in the simulator. The sum of the obstruction weights on the line segment between transmitter and receiver is taken into account when calculating the PRR.

Buildings are also represented using images, one for each floor. Different wall materials are drawn on the floor-plan with different pixel intensities, which are again mapped to obstruction weights that influence PRR calculations. Transmissions between building levels are also affected by the thickness and construction of the floor material. The channel model is easy to understand as it operates at the level of PRR, which is a metric that most applications will be directly concerned with. PRR data from a network deployment is easy to capture and can be used to refine the channel model over time.

ACKNOWLEDGMENT

This research was supported by the grant entitled Research Consortium in Speckled Computing funded by the Scottish Funding Council as part of the Strategic Research Developmen Programme (R32329), and the UK Engineering and Physical Sciences Research Council (EPSRC) as part of the Basic Technology Programme (C523881).

REFERENCES

[1] M. Zuniga and B. Krishnamachari, "Analyzing the transitional region in low power wireless links," in *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, pp. 517–526, IEEE, 2004.

[2] J. Molina-Garcia-Pardo, A. Martinez-Sala, M. Bueno-Delgado, E. Egea-Lopez, L. Juan-Llacer, and J. Garcia-Haro, "Channel model at 868 mhz for wireless sensor networks in outdoor scenarios," in *Proc. International Workshop on Wireless Ad-Hoc Networks (IWWAN 2005)*, Citeseer, 2005.

[3] G. Zhou, T. He, S. Krishnamurthy, and J. Stankovic, "Impact of radio irregularity on wireless sensor networks," in *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pp. 125–138, ACM, 2004.

[4] J. Robinson, R. Swaminathan, and E. Knightly, "Assessment of urban-scale wireless networks with a small number of measurements," in *Proceedings of the 14th ACM international conference on Mobile computing and networking*, pp. 187–198, ACM, 2008.

[5] O. Chipara, G. Hackmann, C. Lu, W. Smart, and G. Roman, "Practical modeling and prediction of radio coverage of indoor sensor networks," in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pp. 339–349, ACM, 2010.

[6] R. Smith, "SPOTWorld and the Sun SPOT," in *Proceedings of the 6th international conference on Information processing in sensor networks*, pp. 565–566, ACM, 2007.

[7] M. Klepal, R. Mathur, A. McGibney, and D. Pesch, "Influence of people shadowing on optimal deployment of WLAN access points," in *Vehicular Technology Conference, 2004. VTC2004-Fall. 2004 IEEE 60th*, vol. 6, pp. 4516–4520, IEEE, 2004.

[8] J. Wilson and N. Patwari, "Radio tomographic imaging with wireless networks," *IEEE Transactions on Mobile Computing*, pp. 621–632, 2009.

Two-way real time fluid simulation using a heterogeneous multicore CPU and GPU architecture

José Ricardo da S. Junior*, Esteban Clua*, Anselmo Montenegro*,
Marcos Lage*, Cristina Vasconcellos * and Paulo Pagliosa†
*Computation Institute
Federal Fluminense University, Niteroi - Brazil
Email: jricardo,esteban,anselmo,mlage,crisnv@ic.uff.br
†Mato Grosso Federal University - DCT - Brazil
Email: pagliosa@dct.ufms.br

***Abstract*—Natural phenomena simulation, such as water and smoke, is a very important topic to increase real time scene realism in video-games. However, the computational fluid simulation is an expensive task since we must numerically solve the Navier-Stokes equations. Additionally, an immersing simulation requires interaction between the flow and the objects in the scene, increasing even more the computational work.**

In this paper we propose an heterogeneous multicore CPU and GPU scalable architecture for fluid simulation with two-way interaction with solid objects. We also show the impact of this architecture over GPU and CPU bounded simulations and present results that can reproduce complex fluid behavior in real time applications like games.

I. Introduction

Realism in video-games is not only a matter of perfect graphics, but also includes the search for real behaviors and physics. While many improvements had been done for dynamic rigid bodies elements, there are still many aspects to be investigated in fluids simulation research. This is an important topic for the game industry, since real aerodynamics effects or liquids behaviors are present in almost any title that simulates real environments.

In order to achieve real immersion, interaction between fluid and rigid bodies placed in the simulated ambient must be considered. However, the collision between different objects with the fluid and the approximation of the physical forces coming from these interactions require time consuming computations. Most times those effects are neglected for video-games, since the physics simulation is only one of many tasks that must be handled by the engine.

The first real time fluid simulations were performed in CPU [1]. Most of these works did not consider the interaction between fluid and scene objects in order to achieve interactive frame rates. Later, with the processing power provided by the GPUs, other phenomena could be coupled to real time fluid simulations. The reader can find methods that consider two-way interactions between fluid and its surrounding objects in the woks [2], [3].

For years, CPUs processors are presenting consistently delivered increase in clock rates, following Moore's law, allowing single-threaded programs to execute faster. However, to manage heat dissipation, the newest CPUs are being designed as multi-core chips, requiring programs to be coded in multi-threaded to take full advantage of these new hardware. As an example of this tendency we can cite the newest Intel's project: a new architecture called "Knights Corner", that allows 50 processing cores in a single chip.

Most works in games develop strategies in which the steps of simulation are performed exclusively by the CPU or GPU. In cases where the GPU is used, the CPU remains idle on the simulation computation, being responsible only for coordinating the graphic processor tasks. Good engine architectures uses this idle time to other tasks, such as artificial intelligence, culling or data structures manipulations. However, as modern CPUs have many cores, most of them remain stranded even with these tasks distribution. In many cases, this waste of time is due to the long time required to transfer data between CPU and graphic device.

In this paper we present a new and efficient architecture for simulating fluid and rigid bodies using a heterogeneous multicore CPU and GPU system, which allows two-way interaction between them. At the same time, we fill the lack of GPU bounded architecture for collision detection, which in many cases processes all elements in the scene at each frame without making any presort of which objects really need to be processed. For performance increase, memory independent regular grids are used for fluid, static and dynamic rigid bodies with an efficient communication between them. As far as we know, we did not find any related work that uses a heterogeneous system for performing this kind of simulation. Additionally, this architecture is automatically scalable with the growing in GPU and CPU cores number. In this work, the fluid simulation in performed through the Smoothed Particle Hydrodynamics (SPH) method, a meshfree particle Lagrangian approach. It is important to say that our heterogeneous architecture of simulating fluid with two-way rigid body interaction increased performance over the last GPU bounded simulations.

The remainder of this paper is organized as follows. After referring to related fluid simulation works, in section II, we describe the fluid and rigid body governing equations in sections III and IV respectively. Next, in section V, we present the acceleration data structure that was developed. In section VI, we introduce our heterogeneous architecture for multicore

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

CPU and GPU. The results are shown in section VII. Finally, in section VIII we present the conclusions of the paper.

II. Related work

The first physical simulation using an Eulerian grid-based approach was originally proposed by Foster and Metaxas [4], [5]. They proposed to solve the full 3D Navier-Stokes equations in order to recreate visual properties of dynamic fluids. In [6], Stam simulated dynamic gases using a semi-Lagrangian integration scheme that achieves unconditional stability using artificial viscosity and rotational damping. Foster and Fedkiw [7] extended the technique to liquids using both a level-set method and particles inside the liquid. Enright et al. [8] added particles outside the fluid for free surface tracking.

In order to allow two-way rigid body interaction, Takahashi et al. [9] presented a simple method to couple fluids and buoyant rigid bodies using regular grids and a combination between the volume of the fluid and Cubic Interpolated Propagation methods. Génevaux et al. [10] used marker particles for free surface representation and to perform interaction with deformable rigid bodies. In [11], the authors used a moving grid to simulate fluid, which allowed to compute the fluids properties anywhere in space.

After the introduction of particle systems by Reeves [12], simulation of natural phenomena using meshless methods started to be proposed. Fluid simulations using the Smoothed Particle Hydrodynamics (SPH) method were proposed by Desbrun and Gascuel [13] to reproduce deformable objects. This approach was latter extended in [14], allowing lava simulation through viscosity and temperature coupling.

Müller et al. [15] used the SPH method to perform fluid simulation in real time. Latter, the authors extend their work in [16] by including fluid interaction with rigid bodies to simulate virtual surgery using a Gaussian Quadrature to distribute ghost particles on rigid bodies surfaces, which are responsible for generating repulsive forces.

Kipfer and Westermann [2] used SPH to simulate fluid flows over deformable terrains. These simulations were performed in GPU using a shader based approach, without considering collision with rigid bodies. Kurose and Takahashi [3] proposed an approach to simulate fluids and rigid bodies with two-way interaction between them using SPH in GPU. The rigid bodies were represented by polygons and they solved a Linear Complementary Problem (LCP) to compute the collision forces between fluid and solids.

III. Fluid simulation

In order to perform fluid simulations, we need to solve the system of differential equations:

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v}.\nabla\mathbf{v}\right) = -\nabla p + \rho\mathbf{g} + \mu\nabla^2\mathbf{v}, \tag{1}$$

and

$$\frac{\partial \rho}{\partial t} + \nabla.(\rho\mathbf{v}) = 0, \tag{2}$$

which are known as Navier-Stokes equations for Newtonian incompressible flows. In these equations, ρ represents the fluid's density, $\mathbf{v}$ the velocity field, p the pressure field, $\mathbf{g}$ the resultant of external forces and μ the fluid's viscosity.

Equation (1) is known as *equation of motion* and states that changes in the linear momentum must be equal to the sum of all forces acting in the system. The convective term $\mathbf{v}.\nabla\mathbf{v}$ represents the rate of change of the velocity field in a fluid element while it moves from one position to another. The convective term is null in Lagrangian methods since the particles used on the fluid discretization follows the flow.

Equation (2) is known as *continuity* or *mass conservation* equation and states that in the absence of sinks and sources the amount of mass in the system must be constant. For particle based methods this equation is unnecessary since each particle carries a constant quantity of mass [15].

In this paper, the Navier-Stokes equations are solved using the SPH method that was introduced by Lucy [17] and Gingold and Monaghan [18] to perform simulations of astrophysical problems. The SPH is a meshless Lagrangian method that evaluates (anywhere in space) the field quantities defined only at discrete set of particles using a compact support, radial and symmetrical smoothing kernel [19].

The evaluation of a continuous scalar field $A(\mathbf{x})$ is achieved by calculating a weighted summation of contributions for all particles $i \in [1...N]$, with position $\mathbf{x}_i$, mass m_i and additional attributes A_i using

$$A(\mathbf{x}) = \sum_j m_j \frac{A_j}{\rho_j} W(\mathbf{r}, h), \tag{3}$$

where ρ_i is the density of particle i, $\mathbf{r} = \mathbf{x} - \mathbf{x}_j$ and $W(\mathbf{r}, h)$ is the smoothing kernel. To compute the density ρ_i of a particle i, we rewrite equation (3) as follows:

$$\rho_i = \rho(\mathbf{x}_i) = \sum_j m_j W(\mathbf{r}, h). \tag{4}$$

The kernel function $W(\mathbf{r}, h)$ must have compact support, i.e. $\int W(\mathbf{r}, h)dr = 1$ and $W(\mathbf{r}, h) = 0$ for $|\mathbf{r}| > h$. According to [20], the value of h must be chosen in order to maintain the number of neighbors inside a particle support approximately 5, 21 and 27 in one, two and three dimensions, respectively.

The gradient and Laplacian of a smoothed attribute function $A(\mathbf{x})$ is the gradient and Laplacian of the kernel function:

$$\bigtriangledown A(\mathbf{x}) = \sum_j m_j \frac{A_j}{\rho_j} \bigtriangledown W(\mathbf{r}, h), \tag{5}$$

$$\bigtriangledown^2 A(\mathbf{x}) = \sum_j m_j \frac{A_j}{\rho_j} \bigtriangledown^2 W(\mathbf{r}, h). \tag{6}$$

In SPH, the pressure field is computed using an equation of state that is a modification from the ideal gas law, as suggested by Desbrun [13]

$$p_i = k(\rho_i - \rho_0), \tag{7}$$

where k is the stiffness constant of the fluid and ρ_0 is its rest density. Finally, the acceleration of particle i is computed as the sum of pressure, viscosity and external forces, being the last one the sum of rigid body interaction and gravitational forces.

IV. Rigid body model

Performing rigid body simulation in games, in many cases, can be summarized as the process of applying constant and external forces to the rigid body's center of mass and calculating its the collisions with other scene objects. The collision checking is sometimes one of the most expensive task during rigid body simulation, requiring calculations of forces and others information such as normal of the contact point and depth of interpenetration between the colliders. In general, performing collision detection between rigid bodies is done through the usage of bounding elements or a polygon's mathematical function representation. Collision detection using simple bounding surface may be much faster than using a polygonal collision. In order to achieve high performance, the first step during collision detection is made by using a coarse representation of the whole rigid body in a bound volume that can be evaluated mathematically, such as its bounding sphere or bounding box. After this stage, normally called the broad phase, rigid bodies that have its geometry overlapped go through a refined collision stage, known as the narrow phase, where a more accurate and computer expensive collision is done, normally using its own rigid body's geometry [21]. The usage of a SIMT (Single Instruction Multiple Thread) architecture, such as CUDA, allows collision detection to be performed even faster, assuming the fact that only one function is used for collision detection.

In this paper we discretize rigid body's polygons into a set of equally radius spheres. This way, performing collision detection between rigid bodies and fluids is a matter of checking collision between spheres, as fluid is also represented in the simulation by a collection of equally radius spheres. For this discretization, we use a modified version of the depth peeling algorithm [22], originally used for rendering transparent polygons. The technique has also been applied to various other purposes, such as collision detection [23] and polygonal discretization [24]. In this paper, this task is performed during a pre-processing step, before the simulation actually starts. The results are presented in Figure 1. It is important to notice that these particles are used only during collision detection with fluid and other rigid bodies in the scene. Collision response is made through repulsive forces using Discrete Elements Method (DEM) as explained in section VI-D. The resolution of the spheres may be controlled and adjusted taking into consideration the importance of the object and the total amount of available process.

V. Acceleration structure

Fluids and rigid bodies are represented using a set of particles that interact with each other. This interaction needs to be performed frequently for fluid simulation, as each particle needs to find its neighborhood particles for calculating variables like pressure and density, according to the SPH method. It's also necessary to retrieve the particles neighborhood relation during the computation of fluid-rigid body and rigid body collisions.

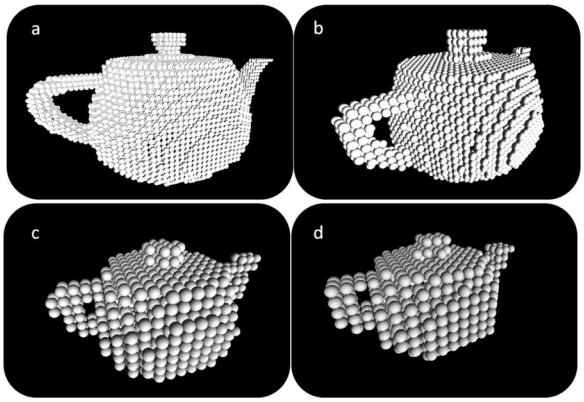


Fig. 1. Teapot processing using variable radius parameter: (a) 0.3 radius with com 7337 spheres; (b) 0.5 radius with 2674 spheres; (c) 0.9 radius with 600 spheres; (d) 1 radius with 668 spheres.

This operation has complexity of $O(n^2)$ for a collection of n particles using a brute force method, being an expensive operation, even for a small set of them. To avoid this time complexity, this paper employs an acceleration structure based on hash tables for locating nearby particles, which also allows the usage of an unbounded world. This acceleration structure requires a predefined number of slots, called buckets. Each of these buckets has two variables, indicating the starting and ending offset in the array containing the index for a particle, as shown in Figure 2 for an eight bucket hash grid. A bucket that does not have any particle is set with a special flag, to avoid its wrong computation during simulation.

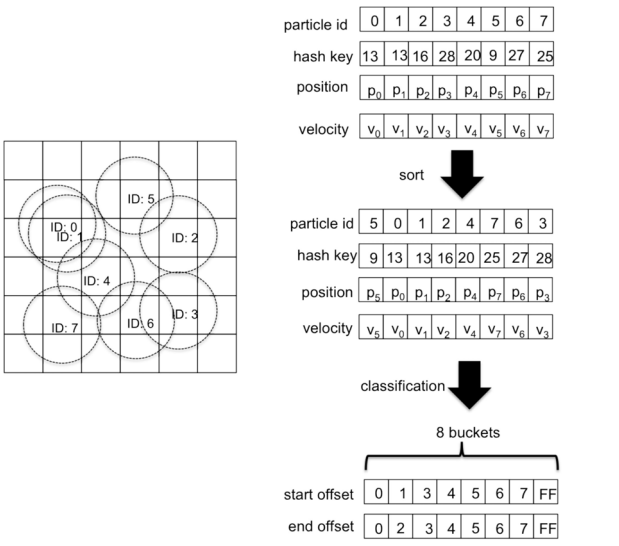


Fig. 2. Process of generating an acceleration data structure based on a hash table.

Before the hash table processing, a preliminary operation

produces a hash key for each particle by using the absolute position of the particle through the use of the algorithm proposed by Teschner et al. [25]. This algorithm receives *p1, p2 and p3*, which are the greatest prime numbers used to minimize hash key conflicts (chosen in our tests as *73856093*, *19349663* and *83492791*, respectively). It also receives the parameter *cell_size*, that represents the imaginary grid's cell size.

We introduced a new parameter in the proposed algorithm in order to avoid an observed problem. Following the original algorithm the same hash key is produced for particles located at symmetrical positions in the world, causing unnecessary data processing. In our proposal a new parameter named *world_limits* is included in the hash key generation formula. It represents the world's bounding box, calculated at each time step. The algorithm with our modifications is presented in Algorithm 1.

Algorithm 1: Algorithm for hash key genenation.

Input: float3 pos, float3 cell_size, int num_buckets, float3 world_limits
Output: unsigned int hash_code
int x ← (int) ((pos.x + world_limits.x) / cell_size.x);
int y ← (int) ((pos.y + world_limits.y) / cell_size.y);
int z ← (int) ((pos.z + world_limits.z) / cell_size.z);
hash_code ← ((x * p1) xor (y * p2) xor (z * p3)) mod num_buckets;

Following, after each particle's hash key calculation, it is necessary to sort these particles based on its calculated hash key. This operation is done in GPU by using the radix sort algorithm [26], presented in CUDPP library [1].

In our proposal, fluid's and rigid body's particles do not have to have the same radius. To achieve this possible radius difference, our work employs more than one hash table, each localized in independent memory spaces. For the fluid's particles case, the kernel radius K_r is used for the size of the imaginary cell size. Derived from the uniform grid adopted in our proposal, only 27 buckets are processed during fluid's particle processing (three grid cells in each dimension).

For the dynamic rigid body case, as previously detailed, the simulated object is discretized into a set of particles that are also manipulated using a hash table. This hash table is independent of the one used by the fluid's particles. Thus, it is located at a different memory space from the former. Knowing that all the rigid bodies are discretized into a set of particles of R_r radius, the same algorithm is used for calculating each particles hash key (using a cell_size of R_r).

Finally, a third hash table is used for sorting static rigid body's particles. Our decision of splitting the simulated rigid bodies into separated hashes is motivated by the fact these particles do not move nor rotate during the simulation, allowing our solution to process them offline and only once, before the simulation actually starts. Hence, the particles of static rigid bodies are inserted into this hash table at the initialization step with its absolute position in space.

In order to compute the fluids and rigid body interaction, for instance, to process collisions between them, it is necessary to access data located at different hash tables. Our solution to this problem was to create a mapping function between those data structures. The function is responsible for mapping a given particle's position from one hash table into a different one.

The mapping function is based on a relation between the properties of the two hash table we want to map. Supposing that X_{to} is the hash table's cell size where a particle is mapping to; and X_{from} is the particle's hash table cell size, we define the transform ratio as $T_r = X_{to}/X_{from}$.

VI. Heterogeneous architecture

A load-balance CPU-GPU is generally made for generic tasks [27]. In this case, these tasks running in parallel must be independent of each other in order to avoid problems of data corruption over multiple threads. This fact makes it difficult to use the CPU and GPU together for solving a simulation. Also, shared processing between CPU and GPU may slow down the overall system performance more than using only GPU due to the data memory exchange that may occur.

In this section we present a new heterogeneous architecture that uses multiple CPU cores and GPU for fluid and rigid body simulation, at same time allowing two-way fluid and rigid body interactions. In this heterogeneous architecture we also focus on minimizing data exchange and concurrent data access during the simulation.

A. CPU-GPU Load balance strategy

Simulating fluid and rigid bodies requires the execution of some ordered tasks and data interchange between them for achieving two-way interaction, as can be seen in Figure 3. This dependency may cause overall system performance degradation in the case when one of these steps takes too long to process data necessary by a dependent stage. To minimize this bottleneck, these tasks need to be well distributed between the GPU and CPU cores in an heterogeneous system, considering the parallel GPU's power compared to the CPUs.

Usually, fluid simulation considering two-way rigid body interaction using some sort of rigid body discretization in GPU is made by performing operations in all of these particles at each frame [28], [29], [3], [24]. Considering for example, *400* rigid bodies discretized into *76* particles each, operations such hash key generation and collision detection need to be performed for *30400* particles. Additionally, fluid's particles also needs to be processed, which normally is in a high number for obtaining correct numerical values in SPH [20].

To avoid unnecessary processing, we developed an architecture, presented in Figure 4, that process rigid body's particles only when they collide with each other or with the fluid's particles. Thus, when no collision between rigid bodies and fluid occurs, their simulation can proceed independently of each other without any data exchange. On the other hand, in case of

[1] Available at http://gpgpu.org/developer/cudpp

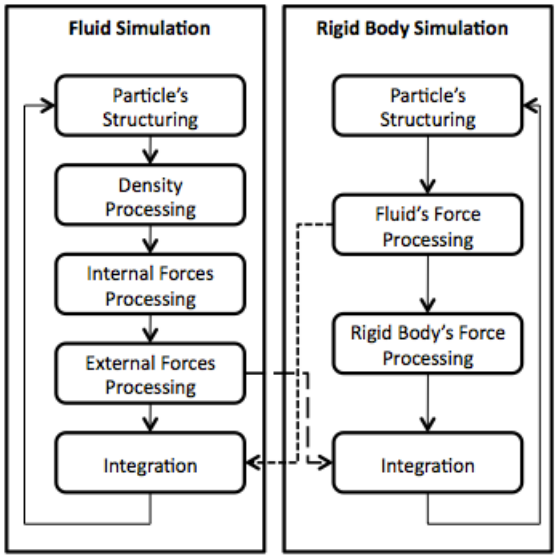


Fig. 3. Necessary stages for performing fluid and rigid body simulation and its dependency showed by the dashed lines.

collision, these exactly particles must take further processing for a more accurate collision and response calculation.

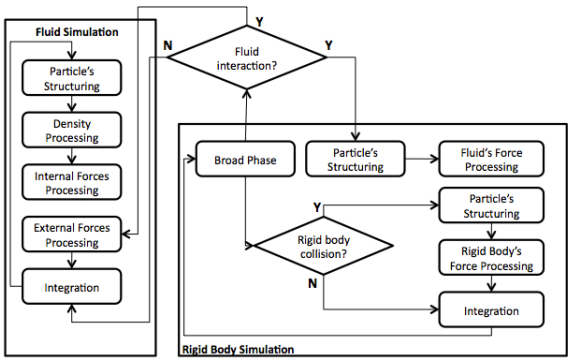


Fig. 4. Implemented model for fluid and rigid body simulation with two-way interaction.

In our proposed heterogeneous architecture, tasks are distributed between CPU and GPU according to Figure 5. As it can be seen in this figure, the GPU is responsible for processing all fluid simulation stages as well as rigid bodies that have collided with other rigid bodies or the fluid. Our decision is driven by the fact that during collision, a high number of particles need to be processed in the narrow phase for accurate collision detection. In the architecture, the CPU is responsible for the broad phase and the integration step of rigid bodies that had no collision with other rigid bodies nor with the fluid, being processed in parallel using multiple CPU cores during fluid's particles spatial subdivision.

B. Broad phase

The effectiveness of this heterogeneous architecture and simulation relies on processing only particles that need to

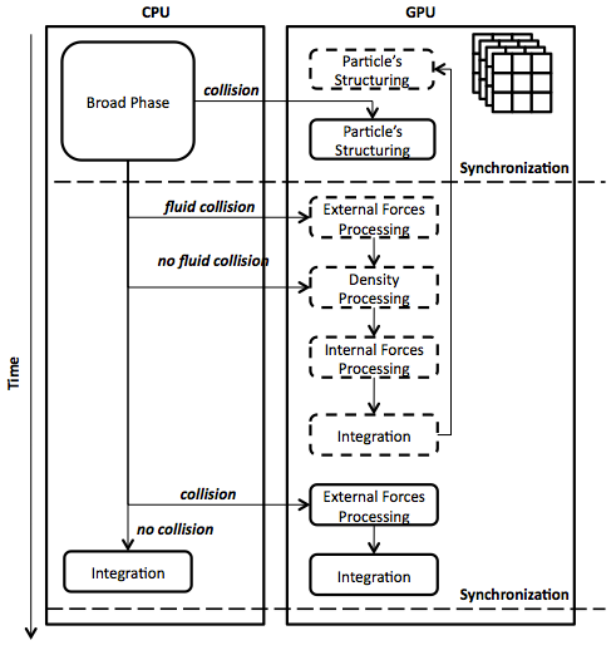


Fig. 5. GPU-CPU task distribution. Dashed blocks represent fluid's tasks while full blocks represent rigid body's tasks.

be processed for accurate collision and two-way interaction with the fluid. During simulation, the following collision possibilities can be observed: rigid body with fluid, rigid body with other rigid body only, and rigid body with rigid body and also fluid. By detecting these types of collision, the two-way interaction between rigid body and fluid and its associated tasks can be performed only when they are necessary.

The broad phase collision detection is made between fluid and rigid bodies by using an *axis aligned bounding box (AABB)* approach. For this, all the available CPU cores are used through the *OpenMP* library, with allows CPU multi-thread programming. During this, each CPU core is responsible for performing the collision detection in a subset of all rigid bodies in the scene and the fluid's volume using its bounding box. In case of collision, a flag is checked in a flag array, indicating if this collision was with a rigid body, fluid or both. For more clarity, this process is shown in Figure 6, where the dashed blocks store the number of collisions detected on each CPU core.

At the end of this process an array containing the potential rigid body colliders whose bounding volume overlapped and a flag array indicating the type of collision is generated. Using these data, a position, an offset and a count array is created independently for the ones that collided with fluid and rigid body. These arrays are responsible for storing the rigid body's relative particles position, the offset index for the starting particle set of each rigid body's in the position array and the particle number of each rigid body, respectively, as presented in Figure 7. Processing interaction between rigid bodies and fluid is only necessary in this array set instead of all elements presented in the simulation, as is commonly done. Rigid bodies

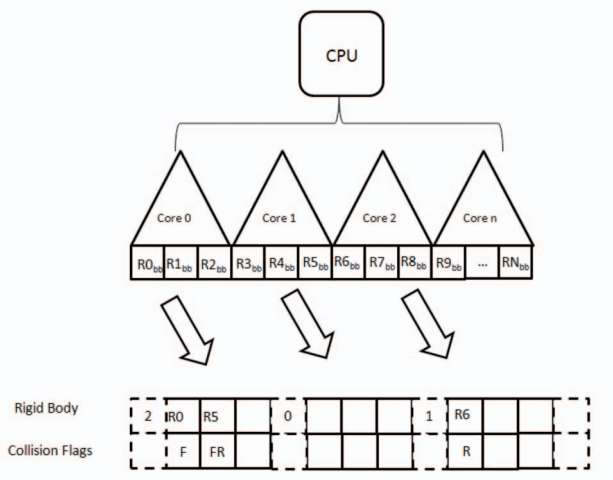


Fig. 6. Performing *the broad phase* using multiple CPU's cores. Rk_{bb} represents the *k-th* rigid body's *bounding box*. *F* flag represents fluid collision while *R* flag represents rigid body collision.

that had any collision are processed by CPU's cores just by integrating its center of mass.

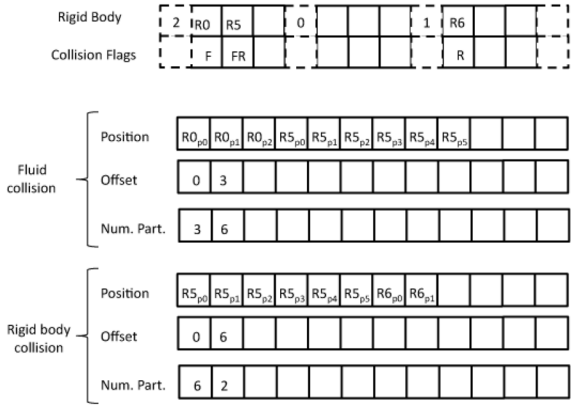


Fig. 7. Data array generated after the broad phase stage in CPU. Rigid body's particles that collide with fluids are stored independently of the ones that collided with fluid.

C. Fluid and rigid bodies collision

After the broad phase step, rigid bodies' particles that collided with the fluid and other rigid bodies are stored in two distinct arrays for the narrow phase processing. Hash key generation and sorting are made only for these particles using its own regular grid, being the narrow phase only processed in these selected particles. In the narrow phase, collision detection is made by using the mathematical sphere equation, which informs if collision occurs and the depth of interpenetration.

D. Forces calculation

During fluid and rigid body simulation, external forces coming from collision as well as the ones caused by gravity are computed. Also, two-way coupling between rigid body and fluid is only computed for the rigid body's particles that had collided in the broad phase step, allowing the fluid to continue its own processing when no collision occurs.

These forces are calculated using the discrete elements method (DEM), which is used for simulating granular materials [30] like sands. The repulsive force f_{ij}, acting on particle i through interaction with particle j is computed using a spring force $f_{i,s}$ and a damping force $f_{i,d}$.

The forces generated during particles interaction need to obey third Newton's law, which states that for every action there is a reaction of the same magnitude but with opposite direction. So, the same approach presented here is used for fluid/rigid body and rigid body/rigid body interaction, storing each force in its respective particle.

E. Integration

After forces are computed for each particle, it is necessary to integrate them to compute the acceleration, velocity and, finally the position. Integration in this paper uses the explicit Eulerian approach.

For fluid's particles, the internal and external forces previously computed are integrated for each particle, and its velocity and position variation is calculated.

For rigid bodies, another approach must be taken due to its discretization. During collision, each particle stores its external force and torque amount coming from interaction with other fluid or rigid bodies' particles. At the end, particle's forces and torque must be added and the resulting force and torque applied directly in the rigid body's center of mass. Our approach uses a scan operation [31] in GPU for this task. However as rigid body's particles forces and torques are stored in a unique array, performing a scan would add all forces from all rigid body's particles that had collided. To solve this, a segmented scan operation [32] is used instead of the scan defined by the CUDPP library. The segmented scan employs an auxiliary array indicating the start of each segment in the array to be processed, doing the addition over all particles from different subsets of the whole array.

An observed limitation in this library for this operation was that it operates only on primitive types such float and integers. Forces normally are represented using a vector type for best representation. For solving this problem, we augmented this library for allowing vector types to be used in the segmented scan operation. With this modification, doing a backwarded inclusive segmented scan executes the segment addition, storing its results in the first element of the segment, which result can be applied in the rigid body's center of mass.

VII. Results

This section presents the results obtained from our heterogeneous multi-core CPU and GPU architecture. For these tests, a PC equipped with an Intel Core 2 Quad Q6600 using 4 GB of RAM and a NVidia Tesla C1060 for processing and a NVidia 8400 GS with 512 DDRAM for visualization was used. Simulations tests with different configuration were performed. During all simulation tests, the *teapot* model was

discretized into *76* particles. Fluid rendering is done in screen space through applying a bilateral filter in sphere's normals.

First, Table I shows the simulation of rigid body and fluid made entirely in CPU and GPU. The column labeled FPS represents the *frames per second* which measure a time necessary to update and render the simulation. Additionally, it also presents an IPS column that represents the *interaction per second*, which measures the time of updating the simulation, without considering rendering time. *Speedup* is measured by the relation of column X^1 over Y^2.

TABLE I
RESULTS OF FLUID AND RIGID BODY SIMULATION WITH TWO-WAY INTERACTION BETWEEN THEM. RB: NUMBER OF RIGID BODIES; CF: TOTAL OF FLUID'S PARTICLES; TPS: TOTAL OF PARTICLE'S SYSTEM.

			GPU		CPU		
CF	RB	TPS	FPS	IPS[1]	FPS	IPS[2]	Speedup
4096	200	19296	23.3	82.0	0.6	0.6	136.66
4096	400	34496	13.7	36.5	0.3	0.3	121.66
8192	200	23392	14.5	40.1	0.5	0.5	80.20
8192	400	38592	9.7	22.2	0.3	0.3	74.00

The result using the new architecture of heterogeneous multi-core CPU and GPU for processing fluid and rigid body simulation with two-way interaction between them is presented in Table II and its graph in Figure 8. The increased performance of our heterogeneous architecture over GPU bounded simulation is shown in Table IV.

TABLE II
RESULT OF FLUID AND RIGID BODY SIMULATION IN THE HETEROGENEOUS ARCHITECTURE WITH TWO-WAY INTERACTION BETWEEN THEM. RB: NUMBER OF RIGID BODIES; CRP: TOTAL OF RIGID BODY'S PARTICLES; CF: TOTAL OF FLUID'S PARTICLES; TPS: TOTAL OF PARTICLE'S SYSTEM.

				Heterogeneous	
CF	RB	CRP	TPS	FPS	IPS
4096	200	15200	19296	28.2	290.2
4096	400	30400	34496	17.9	238.6
8192	200	15200	23392	17.9	120.5
8192	400	30400	38592	12.9	105.6

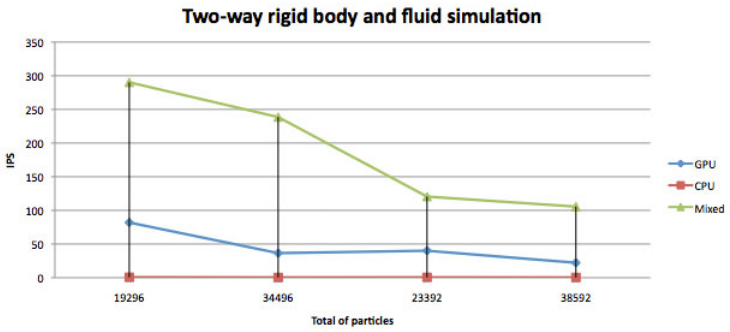


Fig. 8. Graph of rigid body and fluid simulation with two-way interaction in CPU, GPU and heterogeneous (GPU e CPU) system.

In Figure 9 a simulation screen is shown using a total of 80.000 particles, including the fluid's and rigid body's particles. In Figure 10, it's possible to see how heavy objects submerge more in the fluid than light objects.

TABLE III
OVERALL SYSTEM PERFORMANCE INCREASE USING THE HETEROGENEOUS ARCHITECTURE OF MULTIPLE CPU CORES AND GPU OVER GPU BOUND. RB: NUMBER OF RIGID BODIES; CRP: TOTAL OF RIGID BODY'S PARTICLES; CF: TOTAL OF FLUID'S PARTICLES; TPS: TOTAL OF SYSTEM'S PARTICLES.

CF	RB	CRP	TPS	GPU IPS[2]	Het. IPS[1]	Speedup
4096	200	15200	19296	82.0	290.2	3.54
4096	400	30400	34496	36.5	238.6	6.54
8192	200	15200	23392	40.1	120.5	3.0
8192	400	30400	38592	22.2	105.6	4.7

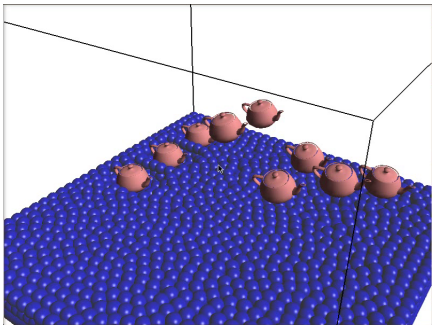


Fig. 9. Fluid and rigid body simulation with two way interaction using our heterogeneous GPU and multicore CPU architecture.

Additionally, to demonstrate the scalability of our heterogeneous architecture, we performed the same tests using an NVidia 9600 GT and the results are presented in Table IV. In this case, processing is automatically redistributed in case of increase in GPU multiprocessor or CPU cores.

TABLE IV
ARCHITECTURE SCALABILITY USING A LOW PROFILE HARDWARE. RB: NUMBER OF RIGID BODIES; CRP: TOTAL OF RIGID BODY'S PARTICLES; CF: TOTAL OF FLUID'S PARTICLES; TPS: TOTAL OF SYSTEM'S PARTICLES.

CF	RB	CRP	TPS	GPU IPS[2]	Het. IPS[1]	Speedup
4096	200	15200	19296	29,5	53,4	1,81
4096	400	30400	34496	22,0	52,6	2,39
8192	200	15200	23392	19,7	28,2	1,43
8192	400	30400	38592	14,9	27,6	1,85

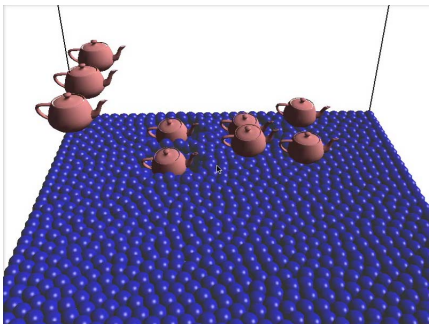


Fig. 10. Two-way physically based rigid body and fluid simulation.

VIII. Conclusions and future works

Our proposed simulation using an heterogeneous system based on multicore CPU and GPU reached a speedup of almost four times the implementations of the most recent full GPU based approaches, allowing the simulation of more complex fluid's behavior in real time in games. The main acceleration factor comes from the fact that many complex and time consuming tasks can be avoided during the simulation processing using a broad phase step. In this case, only rigid bodies that potentially collided with fluid and/or other rigid bodies need further and detailed processing. In most cases these number is very low when compared with the amount of rigid bodies in the scene.

According to the results presented, the proposed architecture can be extended to enable the use of multiple CUDA kernels using the new FERMI GPUs, which allows for more than one kernel to be executed at the same time. At the same time, the heterogeneous architecture could be automatically scalable for the news CPU's design.

Using the rigid body's discretization in this work and the multiple regular grids, level of detail for processing rigid body collision is being developed. In this case, rigid bodies which are in the simulation focus uses a large number of small particles to better perform collision detection and response, while rigid bodies that are not in the simulation focus can use fewer number of big particles to increase the simulation performance during the narrow phase processing. We also state that this could be dynamically adjusted in future works.

Acknowledgements

The author thank all the Computation Institute at Federal Fluminense University for their support. Financial support from CAPES is acknowledged.

References

[1] M. Kass and G. Miller, "Rapid, stable fluid dynamics for computer graphics," in *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1990, pp. 49–57.

[2] P. Kipfer and R. Westermann, "Realistic and interactive simulation of rivers," in *GI '06: Proceedings of Graphics Interface 2006*. Toronto, Ont., Canada, Canada: Canadian Information Processing Society, 2006, pp. 41–48.

[3] S. Kurose and S. Takahashi, "Constraint-based simulation of interactions between fluids and unconstrained rigid bodies," in *Proceedings of Spring Conference on Computer Graphics*, 2009, pp. 197–204.

[4] N. Foster and D. Metaxas, "Realistic animation of liquids," *Graph. Models Image Process.*, vol. 58, no. 5, pp. 471–483, 1996.

[5] ——, "Modeling the motion of a hot, turbulent gas," in *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 181–188.

[6] J. Stam, "Stable fluids," in *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 121–128.

[7] N. Foster and R. Fedkiw, "Practical animation of liquids," in *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 2001, pp. 23–30.

[8] D. Enright, S. Marschner, and R. Fedkiw, "Animation and rendering of complex water surfaces," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 736–744, 2002.

[9] T. Takahashi, H. Ueki, A. Kunimatsu, and H. Fujii, "The simulation of fluid-rigid body interaction," in *SIGGRAPH '02: ACM SIGGRAPH 2002 conference abstracts and applications*. New York, NY, USA: ACM, 2002, pp. 266–266.

[10] O. E. Arash, O. Gnevaux, A. Habibi, and J. michel Dischler, "Simulating fluid-solid interaction," in *in Graphics Interface*, 2003, pp. 31–38.

[11] J. M. Cohen, S. Tariq, and S. Green, "Interactive fluid-particle simulation using translating eulerian grids," in *I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. New York, NY, USA: ACM, 2010, pp. 15–22.

[12] W. T. Reeves, "Particle systems—a technique for modeling a class of fuzzy objects," *ACM Trans. Graph.*, vol. 2, no. 2, pp. 91–108, 1983.

[13] M. Desbrun and M. paule Gascuel, "Smoothed particles: A new paradigm for animating highly deformable bodies," in *In Computer Animation and Simulation 96 (Proceedings of EG Workshop on Animation and Simulation*. Springer-Verlag, 1996, pp. 61–76.

[14] D. Stora, P.-O. Agliati, M.-P. Cani, F. Neyret, and J.-D. Gascuel, "Animating lava flows," in *Proceedings of the 1999 conference on Graphics interface '99*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 203–210.

[15] M. Müller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications," in *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2003, pp. 154–159.

[16] M. Müller, S. Schirm, M. Teschner, B. Heidelberger, and M. Gross, "Interaction of fluids with deformable solids," *Comput. Animat. Virtual Worlds*, vol. 15, no. 3-4, pp. 159–171, 2004.

[17] L. B. Lucy, "A numerical approach to the testing of the fission hypothesis," in *Astronomical Journal, vol. 82*, 1977, pp. 1013–1024.

[18] R. A. Gingold and J. J. Monaghan, "Smoothed particle hydrodynamics - theory and application to non-spherical stars," in *Royal Astronomical Society, Monthly Notices, vol. 181*, 1977, pp. 375–389.

[19] J. J. Monaghan, "Smoothed particle hydrodynamics," in *Annual review of astronomy and astrophysics. Vol. 30*, 1992, pp. 543–574.

[20] G. R. Liu and M. B. Liu, *Smoothed Particle Hydrodynamics: A Meshfree Particle Method; electronic version*. Singapore: World Scientific, 2003.

[21] M. Moore and J. Wilhelms, "Collision detection and response for computer animation," in *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM, 1988, pp. 289–298.

[22] C. Everitt, "Interactive order-independent transparency," NVidia Corporation, Tech. Rep., 2001.

[23] M. Trapp and J. Döllner, "Real-time volumetric tests using layered depth images," in *Eurographics 2008*. The Eurographics Association, 2008, pp. 235–238.

[24] T. Harada, "Real-time rigid body simulation on gpus," in *GPU Gems 3*, H. Nguyen, Ed. Addison-Wesley, 2007, pp. 611–631.

[25] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross, "Optimized spatial hashing for collision detection of deformable objects," in *In Proceedings of VMV'03*, 2003, pp. 47–54.

[26] B. Huang, J. Gao, and X. Li, "An empirically optimized radix sort for gpu," *Parallel and Distributed Processing with Applications, International Symposium on*, vol. 0, pp. 234–241, 2009.

[27] M. Joselli, M. Zamith, E. Clua, A. Montenegro, R. Leal-Toledo, A. Conci, P. Pagliosa, L. Valente, and B. Feijó, "An adaptative game loop architecture with automatic distribution of tasks between cpu and gpu," *Comput. Entertain.*, vol. 7, no. 4, pp. 1–15, 2009.

[28] Y. Zhang, B. Solenthaler, and R. Pajarola, "Adaptive sampling and rendering of fluids on the gpu," in *Eurographics/IEEE VGTC Symposium on Point-Based Graphics*, New York, NY, USA, 2008, pp. 137–146.

[29] ——, "Gpu accelerated sph particle simulation and rendering," in *SIGGRAPH '07: ACM SIGGRAPH 2007 posters*. New York, NY, USA: ACM, 2007, p. 9.

[30] B. K. Mishra, "A review of computer simulation of tumbling mills by dem part i - contact mechanics," in *International Journal of Mineral Processing, Vol. 71(1-4)*, 2003, pp. 73–93.

[31] M. Harris, S. Sengupta, and J. D. Owens, "Parallel prefix sum (scan) with cuda," in *GPU Gems 3*, H. Nguyen, Ed. Addison Wesley, Aug. 2007.

[32] S. Sengupta, M. Harris, Y. Zhang, and J. D. Owens, "Scan primitives for gpu computing," in *Graphics Hardware 2007*. ACM, Aug. 2007, pp. 97–106.

Pseudo-Random Number Generation on GP-GPU

Jonathan Passerat-Palmbach*†‡ Claude Mazel*†‡ David R.C. Hill*†‡
passerat@isima.fr mazel@isima.fr david.hill@univ-bpclermont.fr
*Clermont Université, BP 10448, F-63000 CLERMONT-FERRAND
†CNRS, UMR 6158, Université Blaise Pascal, LIMOS, F-63173
‡ISIMA, Institut Supérieur d'Informatique, de Modélisation et de leurs Applications, BP 10125, F-63177

***Abstract*—Random number generation is a key element of stochastic simulations. It has been widely studied for sequential applications purposes, enabling us to reliably use pseudo-random numbers in this case. Unfortunately, we cannot be so enthusiastic when dealing with parallel stochastic simulations. Many applications still neglect random stream parallelization, leading to potentially biased results. Particular parallel execution platforms, such as Graphics Processing Units (GPUs), add their constraints to those of Pseudo-Random Number Generators (PRNGs) used in parallel. It results in a situation where potential biases can be combined to performance drops when parallelization of random streams has not been carried out rigorously. Here, we propose criteria guiding the design of good GPU-enabled PRNGs. We enhance our comments with a study of the techniques aiming to correctly parallelize random streams, in the context of GPU-enabled stochastic simulations.**

***Keywords*—Stochastic Simulations, GP-GPU, Pseudo-Random Number Generators, Random Stream Parallelization**

I. Introduction

Stochastic simulations are computationally greedy applications. Consequently, domain experts, trying to decrease over and over again their simulation execution time, look for new solutions. For about five years now, we have seen a growing interest in General-Purpose computation on Graphics Processing Units (GP-GPU) through the simulation community. GP-GPU offer a startling amount of computational power at an incredibly low cost, compared to supercomputers. The introduction of this new kind of architecture implies not only the rewriting of the simulation algorithms but also the tools they use. In this article, we have chosen to focus on Pseudo-Random Number Generators (PRNGs), which are the basis of any stochastic simulation.

Sequential PRNGs have been studied for a long time and finding a good quality PRNG to use in a sequential application has not been a problem for more than a decade. A lot of studies have been accomplished to characterize the statistical quality of PRNGs, leading to several tests batteries software. Nowadays, reference test suites are well-known and used to assess PRNGs. But, just like a prime PRNG should not be considered outside the scope of the application it feeds, we should consider different studies according to the domain implied. For instance, cryptographic applications developers ought to base their choice on the NIST test battery [1], whereas simulationists should use TestU01 [2]. These two test batteries can be considered as a standard for their respective domains at the time of writing.

According to [3], [4], a PRNG should perform well on a single processor before being parallelized. Yet, statistical quality is a necessary but not sufficient condition when selecting a PRNG to use in a parallel context: indeed, parallel streams should be independent. Thus, providing high quality random numbers becomes even more difficult when dealing with parallel architectures. We have to take into account the parallelization technique: how will we partition random streams among parallel processing elements (threads or processors)? How will we ensure the independence between parallel streams in order to prevent the involved simulations from producing biased results? The major problem concerning independence between random streams is that no mathematical proof exists to ensure it. However, some studies lay out well-known techniques to spread random streams through parallel applications [3], [5], [6]. They try to ensure the highest independence between random streams using different strategies. We will consider in this article how and whether these techniques could be implemented on GPU.

Apart from the parallelization technique, another point relies directly on the architecture where the involved stochastic simulations run. If we consider a GP-GPU environment, a new difficulty comes into play: harnessing the power of the device requires a rather good knowledge of GPUs. With recent programming frameworks like CUDA (Compute Unified Device Architecture) or OpenCL (Open Computing Language), merely anyone can develop applications for GPUs, but obtaining the announced performance gain implies a higher level of understanding. Most of the work and considerations exposed here rely principally on NVIDIA CUDA solutions. We are also working with the emerging OpenCL standard [7], but the latter is still not robust enough in our opinion. Its current performances are slower than what you could obtain with CUDA [8]. However, we feel that this standard deserves our interest, and should take on an important part of our further work.

In this article, we will name as processing elements the elements effectively computing data in parallel. In a

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

CUDA GP-GPU environment, threads will represent these elements, since this framework relies on a thread level logic referred to as SIMT (Single Instruction, Multiple Threads). The latter abstracts a much more standard designation known as SIMD (Single Instruction, Multiple Data). GPUs are based on this parallel architecture kind. Here, each thread must be given different data that will be computed by an identical operation. In the case of parallel stochastic simulations, we need to furnish an independent stochastic stream to each thread, in order to prevent potential bias that could be introduced otherwise. The questions are then: how are these random streams produced on GP-GPU? And more importantly: how can we ensure that they are independent?

These two main problems led us to think about design guidelines that will hopefully help developers to design better-shaped PRNGs for GPUs. Throughout these few lines, we will focus on random numbers generated and directly consumed on the GPU. In the next sections, some of the cited works have attempted to speed-up generation using the GPU before retrieving random numbers back onto the host. Current CPU-running PRNGs display fairly good performances thanks to dedicated compiler optimizations. For instance, Mersenne Twister for Graphics Processors (MTGP) [9], the recent GPU implementation of the well-known Mersenne Twister [10], is only announced as being 6 times faster than the CPU reference SFMT (SIMD-oriented Fast Mersenne Twister) [11]. Some studies show that the time spent in generating pseudo-random numbers consumes at most 30% of CPU time for some "stochastic-intensive" nuclear simulations [12], but they are very scarce. In view of the small part of the execution time used by most of the stochastic simulations to generate random numbers, it is not worth limiting GPUs usage at the generation task. Therefore, our study is not tied to the parallelization of random number generation algorithms only, and we do not propose any new PRNG in this paper. On the other hand, we share our experience of using stochasticity in GP-GPU-enabled simulations. Considering the previously evoked SIMT model, stochastic simulations with fewer divergent branches in their source code will behave better on GPU. Still, this work does not intend to list examples of simulations that will run efficiently on GPU. Instead, we focus on the parallelization techniques of pseudo-random streams used to directly feed parallel simulation programs running on GPU (applications are called kernels in the CUDA language).

In this study, we will:

- propose GP-GPU specific criteria for PRNGs design;
- suggest requirements for parallelization techniques of random streams on GP-GPU;
- study, according to the previously introduced requirements, the suitability of well-known PRNGs and random streams parallelization techniques for GP-GPU architectures.

Now, let us start with a brief reminder of GPU architecture essentials, so that we can introduce PRNGs implementation on GP-GPU.

II. Random number generation on GP-GPU

The purpose of this section is not to carry out a full survey of GPU-enabled PRNGs propositions in the literature. An attempt to do so can actually be found in [13]. We focus here on PRNGs implementation level on the GPU. The latter cited survey distinguishes three implementation levels, mapped on the CUDA thread hierarchy: threads, blocks of threads and grid of blocks. These strategies directly impact the PRNG implementation, so as the parallelization technique coupled with it. Let us introduce them to understand the technical prerequisites about the two subjects we are tackling in this study.

Obviously, new PRNG algorithms have to take advantage of GPU intrinsic properties. For simplicity purposes, we will briefly introduce the major concepts that rule GPU architectures, that is to say: heterogeneous memory hierarchy and thread organization. Notions described in this paragraph are represented in Figure 1. On the one hand, the thread organization is here to maximize the performances of the application. Threads are bundled into blocks of threads to be affected to one of the Streaming Multiprocessor (SM) of the GPU. SMs can mostly be considered as GPU equivalents to CPU cores. The important point here is that they have their own thread scheduler, championing threads which data are available. Selected threads then run on Streaming Processors (SP): the processing units of SMs. It is here that the matching notion, namely heterogeneous memories, comes into play. Threads can access several memories displaying various capacities, which we will discuss further in Section III. For now, we just need to keep in mind memory areas hierarchy: i.e. the classification of memories based on their quickness and visibility from threads. From the quickest to the slowest, threads can access: registers, shared memory, local memory and global memory. When registers and local memory are dedicated to a single thread, shared memory is visible to all the threads within a block, while every thread can reach global memory.

This memory organization highly affects the PRNG performances. With the first considered implementation level, using a generator per thread, the internal state of the PRNG has to be saved in each thread's local memory. Most of the time, internal states are formed by several elements contained in arrays. Now, CUDA related works, like [14], specify that arrays declared within a thread are stored in the local memory. Although its name indicates a thread scope, please note that local memory is in fact a subset of global memory, and suffers consequently from the same slowness. This area is also allocated to threads when their register set is spent, since registers are available in limited quantities on GPUs.

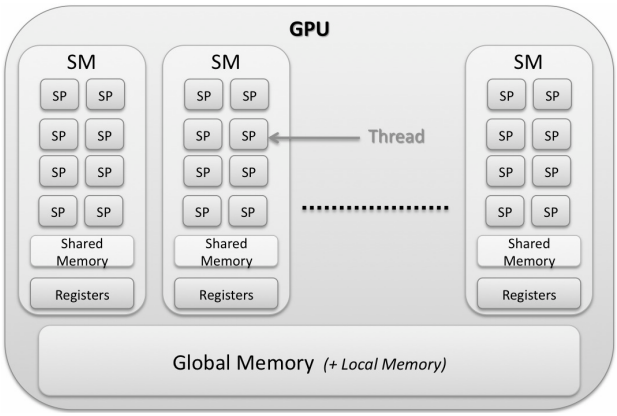


Figure 1. Simple representation of the major elements of a GPU

Equivalently, with the last implementation level, a PRNG for all threads, that is to say a grid-level scope, the global memory is solicited to store the state of the unique PRNG of the application. Each thread draws a number and updates its component of the state in global memory. These two approaches make heavy use of global memory, which has the advantage of being persistent across kernel launches within the same application. Yet, this area is quite slow: it implies a 400 to 800 clock cycle latency because it is not cached [15]. The second implementation scope, the block level, is the only one left to discuss. Every thread in a block can access a shared memory area. PRNG algorithms can consequently store their internal state in this space, enabling every thread of the block to update it. Shared memory is implemented on-chip and is consequently as fast as registers. Thus, PRNGs implemented at a block level will not suffer from the memory latency induced by slow global memory accesses. The three implementation scopes detailed in this section are sketched in Figure 2:

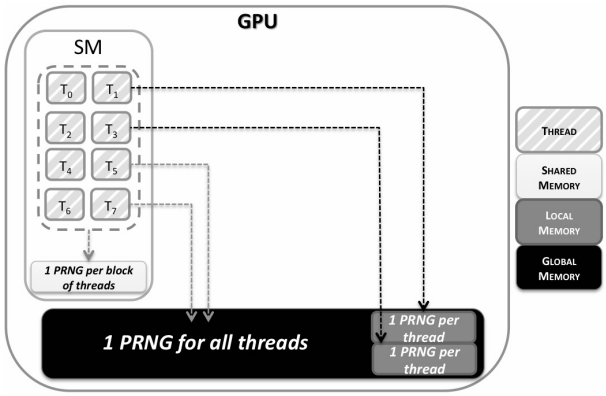


Figure 2. PRNGs implementation scopes and their location in the different memory areas of a GPU

So far, we have described PRNG algorithms, but we also found interesting libraries propositions. We noted two main proposals in this domain: CURAND and Thrust::random. They both aim to provide a straightforward interface to generate random numbers on GPU. Introduced in the latest version, at the time of writing, of the CUDA framework, CURAND [16] has been designed to generate random numbers in a straightforward way on CUDA-enabled GPUs. The main advantage of CURAND is that it is able to produce both quasi-random and pseudo-random sequences, either on GPU or on CPU. The quasi-RNG and pseudo-RNG implemented are respectively Sobol and XorShift [17]. The Application Programming Interface (API) of the library stays the same no matter which kind of RNG is selected and the platform on which the application is run on.

Thrust::random is part of a GPU-enabled general purpose library called Thrust [18]. This open-source project intends to provide a GPU-enabled library equivalent to standard general-purpose C++ libraries, such as STL or Boost. Classes are split through several namespaces, of which Thrust::random is an example. The latter contains all classes and methods related to random numbers generation on GP-GPU. Thrust::random implements three PRNGs, each through a different C++ template class. We find a Linear Congruential Generator (LCG), a Linear Feedback Shift (LFS) [19] and a Subtract With Borrow (SWB) [20], [21]. Although the latter PRNG is mentioned as Subtract With Carry in Thrust::random documentation, Marsaglia's original proposition is known as SWB. In spite of the known flaws laid out by all these generators, the library offers simple ways to combine them into better quality randomness sources, like L'Ecuyer's Tausworthe combined generators [22].

III. GP-GPU specific requirements for PRNGs

As a result of the SIMD parallelism between threads and of their graphics processors legacy, which we superficially discussed in Section II, GP-GPUs are not equivalent to a set of standard processors used in parallel. These particularities introduce some constraints that need to be satisfied if we do not want to see the overall simulation performance dramatically drop. Thus, we will introduce in this section the requirements we find compulsory for a PRNG to run efficiently on a GP-GPU architecture.

The main goal targeted when using GP-GPUs is to obtain greater speed-ups. But this kind of device has not been primarily designed to support general computations and is more inclined to realize some arithmetic operations. Nowadays GPUs display different performances with single and double precision floating point numbers. For instance, the previous generation of NVIDIA supercomputing-dedicated GPU, the Tesla T10, was known to deliver ten times as much computation power when dealing with single precision floating point numbers rather than double. Even if the actual cutting-edge GPU, the NVIDIA new generation Fermi T20, has considerably reduced the gap between these two precisions, it would be wise to remain cautious before using double precision

operations on GPU. Most of the time, single precision floats are sufficient enough to handle random values contained in [0 ; 1[and should consequently be favoured. This proposition leads us to our first criterion: *single precision floating point numbers should be preferred throughout the GP-GPU random number generation algorithm.*

Another legacy of graphics processors is the heterogeneous memory organization. To complete what has previously been said on this subject, let us recall that several memory areas are reachable by threads running on a GPU. These memories capacities, some quick and large, others less so, depend on two characteristics. First, the more threads can reach a memory area, the slower it is. Indeed, registers allocated to a single thread are the quickest memory this thread will be able to communicate with. At the same efficiency level, we find shared memory, reachable by a relatively small amount of threads, all belonging to the same block, and so running on the same core of the GPU. On the other hand, every thread running on the GPU, regardless of which core they are located on, can access a wide memory area, commonly called global memory. Therefore, this latter area is far slower than its previous counterparts. Second, read-only memories are quicker since they can fully benefit from cache mechanisms, contrary to read-write memories. In the light of these memory constraints, it is obvious that GPU PRNGs should be designed with particular attention to sparing costly memory accesses. Commonly, static parameters will take place in read-only areas, whereas dynamic elements such as state vectors will be handled at the thread or thread group level. In a more formal way, the following is another criterion of good design: *the algorithm should be designed in a way to avoid global memory accesses.*

Taking into account the particularities of GPUs and their architecture, we have proposed two new requirements for PRNGs to run efficiently on such devices. They are summed up hereafter:

1) *single precision floating point numbers should be preferred throughout the GP-GPU random number generation algorithm*
2) *the algorithm should be designed in a way to avoid global memory accesses*

IV. GP-GPU specific requirements for random streams parallelization

The literature is full of references describing the profile of what a good usage of parallel PRNGs should be. For example, [3], [4] list requirements that should meet any sequential or parallel PRNG. GPUs are particular parallel architectures. So any PRNG running on this kind of device should, at least, meet the requirements enumerated in the previous references. In this section, we will successively check how these criteria can be adapted to GPU architectures.

Emphasizing parallel PRNG performances, [3] noticed that *each processor should generate its sequence independently of the other processors.* We consider, indeed, that every processing element should have its own stochastic stream at its disposal. This condition must be satisfied first, not only for efficiency, but especially because GPU-enabled stochastic simulation parallelization principles rely on it. First, it is a necessary, but not sufficient, condition to fulfil to ensure a higher independence of stochastic streams feeding different replications of a simulation. Second, considering a single replication, the SIMT parallelism level leads threads to compute their own data sets, including their own stochastic stream. Thus, our first requirement concerning random streams parallelization can be expressed as follows: *each thread should dispose of its own random sequence.*

As we explained previously, GP-GPU programming frameworks offer a thread scope rather than a processor one. The threads in use for GP-GPU propose an abstraction of the underlying architecture. They are concurrently running on the same device and handle their own local memory area. Thread scheduling is at the basis of GPU performances. Memory accesses are the well-known bottleneck of this kind of device. Indeed, running a large amount of threads in turn allows GPUs to bypass memory latency. There should always be runnable threads while others are waiting for their input data. Disregarding the effective number of processors, we theoretically say that the more threads you have, the better your application will leverage the device. Applications need to be written to use the maximum number of threads, but also to scale up transparently when the next GPU generation will be able to run twice as many threads as today. So, in accordance with [3] who advocates that *the generator should work for any number of processors*, our second GP-GPU specific requirement for parallelization techniques of random streams is that *it must be usable for any number of GP-GPU threads.*

Returning to the original requirements, we then find [3] states that *parallel random streams produced should be uncorrelated.* This criterion is related to both PRNG intrinsic properties and to the parallelization technique set up. We previously stated that a PRNG candidate to parallelization should first perform well on a single processor. Thus, we will not take its intrinsic qualities into account here. However, no matter the worth of the used PRNG, the parallelization techniques must be used carefully, as will be shown in the next section. Please note that this requirement is neither affected by GP-GPU architectures nor by programming frameworks. As a result, we will just recall it without modifying its expression.

[3] also noted that *the same sequence of random numbers should be produced for different numbers of processors, and for the special case of a single processor.* Here, we understand that the PRNG output on each processor should not depend on the number of processors used. This point is very important and must be treated carefully when choosing a parallelization technique. For example, in a

distributed environment containing several processors, a scheduler can govern execution. Depending on the scheduler algorithm and on the global system charge, parallel executions of different parts of a simulation might not execute in the same order. In such a case, it is compulsory for the PRNG output to be independent from the order in which simulations parts may run. If this requirement is not met, reproducibility of simulations executions is no longer ensured. We can do this for games, but not for scientific applications. Reproducibility is needed when dealing with stochastic simulations, in order to debug a problematic case raised by a particular random stream for instance. We also think about design of experiments for simulations, where reproducibility is mandatory to isolate parameters variations impact in the results.

In the case of GP-GPU, we find exactly the same problem at the thread level. These entities are also scheduled, not atomically but by bundles. Fortunately, both threads and their bundles own a unique identifier allowing us to distinguish them among executions. Thus, if a parallel random stream is only bound to the unique identifier of a thread, according to our first requirement, output will be reproducible through multiple executions. Therefore, we can obtain the necessary bijective relation between a T_i thread and an SS_i stochastic stream, as stated in Figure 3:

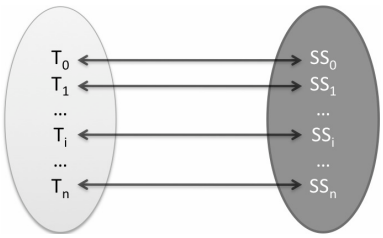


Figure 3. Bijective relation between threads and stochastic streams

Finally, we can write our last requirement in such a way: *when the status of the PRNG is not modified, the sequence of random numbers generated for a given thread must be the same no matter the number of threads and no matter of threads scheduling.*

Let us now sum up the requirements targeting GPUs we highlighted in this part, continuing the enumeration started in Section III:

3) *each thread should dispose of its own random sequence*
4) *the parallelization technique must be usable for any number of GP-GPU threads*
5) *the parallel random streams produced should be uncorrelated*
6) *when the status of the PRNG is not modified, the sequence of random numbers generated for a given thread must be the same no matter the number of threads and no matter of threads scheduling*

V. Do classical random streams parallelization techniques fit GP-GPU well?

This section presents the main techniques used to distribute random streams between processing elements. A more exhaustive and dedicated survey can be found in [6]. We will also study in this section how random streams parallelization techniques can be adapted to GPU architectures, depending on their ability to fulfil the previously introduced requirements (Sections III and IV).

The Leap Frog is the way to partition a random stream like a deck of cards. Random numbers are allocated in turn to processors like cards would be dealt to players. Pragmatically, let each processor hold an i identifier. The latter processing element will build a Y_i substream from an X original random stream such as $Y_i = \{X_i, X_{i+N}, \ldots, X_{i+kN}\}$, with N equal to the number of processors. This technique is not quite adapted to split random streams on GP-GPU since it does not satisfy the last constraint expressed in Section IV. In fact, if the number of threads changes, the subsequence affected to each thread will be different. This situation is shown in Figure 4. Now, the number of threads for an application is bound to the underlying device: GPUs can run a different number of threads concurrently, depending on their architecture generation. As a result, we would not be able to ensure the reproducibility of a simulation from a GPU to another, even if we initialized the PRNG with the same parameters. Furthermore, some bugs induced by random draws might not appear on developer's device, while they would on the user's.

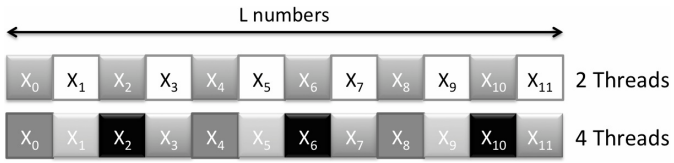


Figure 4. Different threads numbers leading to different random substreams through the Leap Frog method

The solution would be to implement the PRNG at a level with a constant thread number. The CUDA framework handles constant-sized bundles called warps, which always contain 32 threads at the time of writing. They are in fact a subdivision of blocks of threads, and access consequently the same shared memory area. Nowadays, a great number of GPUs do not have enough shared memory to store a PRNG status per warp. However, the newest GPU generations offer larger shared memory spaces that could potentially enable us to use Leap Frog, following this idea.

Apart from any architecture consideration, please note that Leap Frog is not adapted to every kind of PRNGs. [4] has established that this technique could be flawed in some cases, because of spectral tests. We particularly insist on what an unsafe technique it is when used with LCGs.

Serious stochastic simulations should ban simple LCGs since they have structural flaws [23].

The second technique to be referenced in [3] is Sequence Splitting (also known as fixed spacing or blocking). It consists in allocating continuous and equally sized blocks from the original random stream to form substreams. This technique implies the knowledge of how many numbers each thread will consume at most. Indeed, knowing that each thread consumes at most L random numbers, then the first L numbers will be attributed to the first thread, the L following to the second thread and so forth. Following the previous formalism, we have $Y_i = \{X_{iL}, X_{iL+1}, \ldots, X_{(i+1)L-1}\}$. This numbers repartition is described in Figure 5.

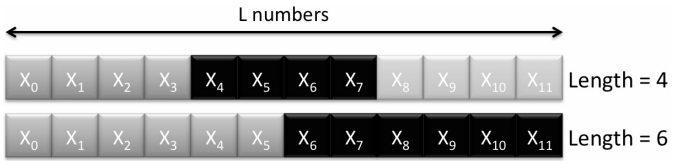


Figure 5. Two random streams parallelizations based upon Sequence Splitting with two different sub-sequences lengths

Efficient Sequence Splitting relies on a particular feature of the PRNG called Jump Ahead, or Skip Ahead. Instead of unrolling the generator, it enables us to compute any given advanced state of the random sequence without processing the previous ones (for example $X_{(i+1)L}$ can be computed directly from X_{iL}). Here, we discern two categories of algorithms. Some PRNGs contain an algorithm able to realize Jump Ahead, thanks to intrinsic properties of the PRNG [24]. This allows us to reach any part of the sequence in equal time, regardless of the destination point. Let this technique be considered as Intrinsic Jump Ahead hereafter. The other solution is to emulate Skipping Ahead: to do so, we have to compute an advanced state by processing step by step the previous ones (for example, starting from $X_{iL}, X_{(i+1)L}$ can be computed by running the PRNG L times in a sequential way in order to get $X_{iL+1}, \ldots, X_{(i+1)L-1}$ and finally $X_{(i+1)L}$). In fact, whichever PRNG you use, you can unfold the sequence to the desired point, and store the state vector at this point in order to be able to load it later. Such a vector is named seed status in [13], since it is able to set a generator in a predefined state. Emulated Jump Ahead can become very costly though: indeed, the further you need to go in the sequence, the more time it takes to compute the seed status.

To conclude on Sequence Splitting, if an intrinsic Jump Ahead algorithm is available for the involved PRNG, Sequence Splitting is a very good approach for GP-GPUs. However, as far as we know there are few ports of algorithms from this family to GP-GPU. On the other hand, Emulated Jump Ahead is not GPU-compliant because statuses computation is a purely sequential operation (we need X_n to compute X_{n+1}). As a consequence, different threads may display different lengths of time for these computations, thus decreasing the SIMD parallelism overall gain. To solve this problem, we propose to pre-compute substreams on the host side, store the seed status at each substream starting point and then transfer all these statuses to the device. In the past, Sequence Splitting was also considered as unsafe by several studies [25], [26], but this remark applies to any PRNGs that showed potential long-range correlations, such as congruential ones.

We have just seen that emulated Jump Ahead led to expensive calculations in order to compute an initial status for each sub-sequence. Another approach based upon probabilities is known as Random Spacing. The idea is to avoid Emulated Jump Ahead cost by choosing random statuses in the base stream, and set them as initial statuses for the resulting substreams as represented in Figure 6. Although Random Spacing spares us from solving the Jump Ahead problem, it should not be used without care. [27] have given the minimum distance between N sub-sequences which status were randomly chosen: it is in average equal to $1/N^2$ multiplied by the period length. More precisely, the probability of overlapping between N sequences of length L, issued from a PRNG of period P is equal to: $1 - (1 - NL/(P-1))^{(N-1)}$. That is equivalent to $N(N-1)L/P$ when $NL/(P-1)$ is in the neighbourhood of 0.

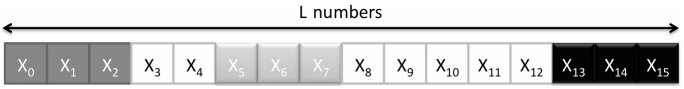


Figure 6. Random Spacing creation of three sub-sequences of equal length but differently spaced from each other

Overlapping risks become sizeable with short period PRNGs. All the PRNGs tested by Pierre L'Ecuyer and Richard Simard in [2] display periods P from 2^{24} to 2^{131072}. Most of them have $\log_2 P$ of the order of a few dozens. Now, given that nowadays longest simulations can consume up to several hundreds of billions of random numbers, ($L = 10^{11}$), a hundred of such replications ($N = 100$) leads to a $N(N-1)L$ of approximately 10^{15}, i.e. 2^{50}. The probability to see an overlapping between two sub-sequences issued from a PRNG of period P far bigger than 2^{50} is negligible. Nonetheless, from 20 LCGs PRNGs tested in [2], 14 laid out an overlapping probability greater than 99.9% (period less than 2^{47}). Only 3 generators have an overlapping probability less than 0.01% ($\log_2 P > 59.7$). In the same way, widely spread PRNGs such as Comblec88 of period $P = 2^{61}$ (Combined LCGs), are on the acceptance borderline for this technique. They are shipped with several renowned software packages though, including: RANLIB, CERNLIB, Boost, Octave and Scilab [2].

As a conclusion, this technique fits GP-GPUs well, but the risk of overlapping between sub-sequences must be evaluated according to the number and the length of the sub-sequences and to the period of the PRNG used. If we

select generators with large periods, such as WELLs and Mersenne Twister, this risk is negligible.

Techniques presented so far tried to split a single stream into several substreams. Another approach consists in using several declinations of the same PRNG: each generator has the same structure and generation mechanism with a unique parameter set, called parameterized status hereafter. By allocating a parameterized status per processor, parameterization ensures our first requirement, namely that each thread should dispose of its own sequence. This situation is described in Figure 7. We always have to remember that this cannot ensure our third requirement: strict independence between parallel random streams.

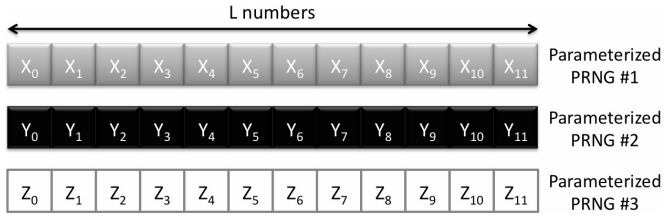


Figure 7. Three parameterized PRNGs producing three highly independent random sequences

Although no mathematical proof can establish this independence, some implementations of Parameterization are safe according to the current state of the art [28]. We especially think to the Dynamic Creator (DC) algorithm [29] coming along with the Mersenne Twister for Graphics Processors (MTGP) PRNG [9], a quality-proven work benchmarked in [13]. DC integrates a unique identifier, which belongs to the parameterized status of MTGP. This identifier becomes a part of the characteristic polynomial of the matrix that defines the recurrence of the PRNG. Two identifiers will consequently lead to two different parameterized statuses. Furthermore, DC ensures that the characteristic polynomials we obtain are mutually prime and the authors assert that the random sequences generated with such distinct parameterized statuses will be highly independent, even if, as mentioned before, this fact cannot be mathematically proven.

This technique displays some constraints making it difficult to port on GPUs. Unfortunately, few PRNGs propose it intrinsically. Some, such as LCGs, are even recognized as bad candidates for Parameterization [30]. In addition, storing seed statuses being already problematic, we can scarcely imagine spending vast amounts of memory to store a parameterized status per thread.

Every technique introduced in this section, presents advantages and drawbacks. Most of them are related to the chosen PRNG. Depending on the application and environment you own, you might be forced to select a PRNG knowing it has some flaws in particular cases. In this way, Table I states PRNG kinds and parallelization techniques that work well together:

Table I
Summary of the potential PRNG/Parallelization technique associations

Technique	***Preferred PRNGs***	***PRNGs to avoid***
Leap Frog	None (disable reproducibility)	Linear generators
Sequence Splitting	Intrinsic Jump-Ahead compliant	Emulated Jump-Ahead
Random Spacing	Large period	Short period
Parameterization	MT family	LCG

VI. Conclusion

We have seen that more and more applications, and especially stochastic simulations, tend to take advantage of recent GP-GPU architectures in order to improve their performance. However GPU computing needs to offer the same tools as other platforms. Good quality PRNGs belong to this category. This paper has shown how difficult it could be to generate good quality pseudo random numbers on GP-GPUs. Indeed, it implies taking into consideration two different domains: GP-GPU programming and PRNG parallelization techniques. Issuing a PRNG that can produce independent stochastic streams when used in parallel is a first hurdle that not all PRNGs can get over. When you have at your disposal one that fulfills this requirement, it has to be ported to GP-GPU. It means that you need to think about a GPU parallelization of your PRNG, if it is not already available.

This paper introduced the main problems that you will be faced with when trying to port your stochastic simulations on GP-GPU. To avoid these difficulties, and above all, errors and performance drops that could result from the use of a hazardous GPU-enabled PRNG, we first proposed PRNGs criteria dedicated to GPUs. Then we defined parallelization techniques requirements in order to make these PRNGs fit GPU peculiarities. They sum up the experience accumulated in our research team concerning GPU-enabled stochastic simulations.

We encountered lots of parameters brought up by PRNGs and GPU programming. As long as it can dramatically impact both the overall performance of the simulation and the quality of its results, it might be a good point to propose a straightforward API to use well-defined PRNGs on GPUs. In this way, libraries laid out in Section II represent in our mind an elegant manner to do so. They allow every user to easily call PRNG functions without wasting time on how parameters should be set. In the same way, they enable advanced users to adjust parameters in their own fashion. As a matter of fact, further works of ours should target this kind of development, embedding PRNGs following the presently established requirements into a GPU-enabled library with a unified API.

Acknowledgements

Special thanks to Susan Arbon, Natacha Vandereruch and Florian Guillochon for their careful reading and useful

suggestions on the draft.

References

[1] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," NIST, Tech. Rep., 2001.

[2] P. L'Ecuyer and R. Simard, "Testu01: A C library for empirical testing of random number generators," *ACM Transactions on Mathematical Software*, vol. 33, no. 4, pp. 22:1–40, August 2007.

[3] P. Coddington, "Random number generators for parallel computers," NHSE, Tech. Rep. 2, 1996.

[4] P. Hellekalek, "Good random number generators are (not so) easy to find," *Mathematics and Computers in Simulation*, vol. 46, no. 5-6, pp. 485–505, 1998.

[5] ——, "Don't trust parallel monte carlo!" in *Proceedinfs of Parallel and Distributed Simulation PADS98*. IEEE, 1998, pp. 82–89.

[6] D. Hill, "Practical distribution of random streams for stochastic high performance computing," in *IEEE International Conference on High Performance Computing & Simulation (HPCS 2010)*, 2010, pp. 1–8, invited paper.

[7] Khronos OpenCL Working Group, "The OpenCL specification," Khronos Group, Specification 1.1, September 2010.

[8] K. Karimi, N. Dickson, and F. Hamze, "A performance comparison of CUDA and OpenCL," http://arxiv.org/abs/1005.2581v2, August 2010, submitted.

[9] M. Saito, "A variant of Mersenne Twister suitable for Graphics Processors," http://arxiv.org/abs/1005.4973, 2010, submitted.

[10] M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionnally equidistributed uniform pseudorandom number generator," *ACM Transactions on Modeling and Computer Simulations: Special Issue on Uniform Random Number Generation*, vol. 8, no. 1, pp. 3–30, January 1998.

[11] M. Saito and M. Matsumoto, "SIMD-oriented Fast Mersenne Twister: a 128-bit pseudorandom number generator," in *Monte Carlo and Quasi-Monte Carlo Methods 2006*, A. Keller, S. Heinrich, and H. Niederreiter, Eds., vol. 2. Springer Berlin Heidelberg, 2008, pp. 607–622.

[12] L. Maigne, D. Hill, P. Calvat, V. Breton, R. Reuillon, Y. Legre, and D. Donnarieix, "Parallelization of monte carlo simulations and submission to a grid environment," *Parallel processing letters*, vol. 14, no. 2, pp. 177–196, 2004.

[13] J. Passerat-Palmbach, C. Mazel, A. Mahul, and D. Hill, "Reliable initialization of gpu-enabled parallel stochastic simulations using Mersenne Twister for Graphics Processors," in *ESM 2010*, 2010, pp. 187–195, iSBN: 978-90-77381-57-1.

[14] D. Kirk and W. Hwu, *Programming Massively Parallel Processors*. Morgan Kaufmann, 2010, iSBN: 978-0123814722.

[15] NVIDIA, *NVIDIA CUDA Programming Guide Version 3.2*, October 2010.

[16] ——, *CUDA CURAND Library*, NVDIA Corporation, August 2010.

[17] G. Marsaglia, "Xorshift rngs," *Journal of Statistical Software*, vol. 8, no. 14, pp. 1–6, 2003.

[18] J. Hoberock and N. Bell, "Thrust: A parallel template library," http://www.meganewtons.com/, 2010, version 1.3.0.

[19] R. Tausworthe, "Random numbers generated by linear recurrence modulo two," *Mathematics of Computation*, vol. 19, no. 90, pp. 201–209, 1965.

[20] G. Marsaglia, B. Narasimhan, and A. Zaman, "A random number generator for pc's," *Computer Physics Communications*, vol. 60, no. 3, pp. 345–349, 1990.

[21] G. Marsaglia and A. Zaman, "A new class of random number generators," *Annals of Applied Probability*, vol. 3, no. 3, pp. 462–480, 1991.

[22] P. L'Ecuyer, "Maximally equidistributed combined tausworthe generators," *Mathematics of computation*, vol. 65, no. 213, pp. 203–213, 1996.

[23] ——, *Encyclopedia of Quantitative Finance*, 2010, ch. Pseudorandom Number Gemerators.

[24] H. Haramoto, M. Matsumoto, T. Nishimura, F. Panneton, and P. L'Ecuyer, "Efficient jump ahead for f2-linear random number generators," *INFORMS Journal on Computing*, 2007.

[25] A. De Matteis, J. Eichenauer-Herrmann, and H. Grothe, "Computation of critical distances within multiplicative congruential pseudorandom number sequences," *Journal of Computational and Applied Mathematics*, vol. 39, no. 1, pp. 49–55, 1992.

[26] A. De Matteis and S. Pagnutti, "Critical distances in pseudorandom sequences generated with composite moduli," *International Journal of Computer Mathematics*, vol. 43, no. 3, pp. 189–196, 1992.

[27] P. Wu and K. Huang, "Parallel use of multiplicative congruential random number generators," *Computer Physics Communications*, vol. 175, no. 1, pp. 25–29, 2006.

[28] M. Mascagni and A. Srinivasan, "Parameterizing parallel multiplicative lagged-fibonacci generators," *Parallel Computing*, vol. 30, no. 7, pp. 899–916, 2004.

[29] M. Matsumoto and T. Nishimura, "Dynamic creation of pseudorandom number generators," in *Monte Carlo and Quasi-Monte Carlo Methods 1998*. Springer, 2000, pp. 56–69.

[30] A. De Matteis and S. Pagnutti, "Parallelization of random number generators and long-range correlations," *Numerische Mathematik*, vol. 53, pp. 595–608, 1988.

A Well-Balanced Time Warp System on Multi-Core Environments

Li-li Chen*, Ya-shuai Lü†, Yi-ping Yao*, Shao-liang Peng* and Ling-da Wu†
*Department of Computer Science
National University of Defense Technology, Changsha, Hunan, China
Email: {chenlili8209, ypyao, pengshaoliang}@nudt.edu.cn
†Academy of Equipment Command & Technology, Beijing, China
Email: yashuailv@nudt.edu.cn wulingda@263.net

***Abstract*—The current trend in processor architecture design is the integration of multiple cores on a single processor. This trend has shifted the burden of improving program execution speed from chip manufacturers to software developers. Thus, in the software domain, one of the research focuses is on modifying software platforms to efficiently utilize the computation resources of multi-core processors. In this paper, we propose a global schedule mechanism based on a distributed event queue to improve the performance of Time Warp system on multi-core systems and give some experiences on the implementation of the shared attribute/state access mechanism based on transactional space-time memory. Furthermore, this paper comprehensively explores how the different design choices and techniques affect the performance of Time Warp system on a multi-core platform by various experiments. Compared with the distributed event queue local schedule mechanism, the experiment results show that the distributed queue global schedule mechanism can effectively decrease rollback rate and balance the workloads at a low event scheduling cost for Time Warp system on multi-core platforms; the STM-based shared attribute access mechanism prominently outperforms the conventional "pull" mechanism on multi-core platforms.**

I. Motivation

With the fast development of VLSI technology, microprocessor architecture has entered the multi-core era. Both the new high performance computation platforms and the new personal computers are using multi-core processors as their CPUs. The fast core to core communication and rich computation resources of multi-core processors may bring new opportunities for analytic simulations. However, without adaptation, the rich computation resources of multi-core processors could not be fully utilized for most software applications (including those analytic simulation applications) that were used on the single-core processors. For those who are not major in computer science, the parallel programming for multi-core systems poses a great burden. Thus, it is the computer scientists' responsibility to develop general-purpose software platforms that can make easy and effective use of multi-core processors.

As a result, various software programming mechanisms have been developed for multi-core systems in recent years. For example, in the programming language domain, OpenMP and MPI are further developed, and new programming languages like X10 [1] are proposed. Thread based parallel libraries like Intel TBB [2], MKL [3] are developed to ease the burden of programming for multi-core processors. In the processor architecture domain, thread level speculate (TLS) [4] and transactional memory (TM) [5] are well studied over the past ten years. These thread level optimistic execution techniques are proposed with the goal of simplifying the transformation from sequential programs to parallel programs and improving the execution performances of programs on multi-core systems. Discrete event simulation has been an active research topic since the beginnings of the computer industry. The most common optimistic PDES protocol is the Time Warp protocol [6]. In most of the protocol implementations, the simulation processes are "bound" to an OS process/thread, and the events are scheduled and executed within that process/thread. This can be considered as a distributed event queue and local schedule mechanism, because the simulation processes and their events can not be transferred from an OS process/thread to another unless there is an additional dynamically transferring mechanism. One problem with this schedule mechanism is that the simulation processes are statically distributed before the simulation, this may lead to poor workloads balancing under certain circumstances.

To solve these problems above, static partitioning strategies and dynamic migrating schemes for workloads balancing have been widely researched. Static partitioning strategies play important roles in some special applications but may not suitable for others. To date, it's been well established that Time Warp dynamic load balancers do well on clusters (e.g., Tropper's XTW system for verilog circuit simulations [7]). But dynamic load-balancing strategies have to perform the following actions: give an algorithm to determine which LPs should be selected to move to other processes/threads; pause the simulation during the selected LPs' transfer before allowing the simulation to proceed, which may lead to waiting cost.

It is believed that the globally event scheduling approach to building Time Warp kernels is the preferred implementation methodology if the absolute best workloads balancing is required. But its potentially overhead on globally scheduling may be too great to benefit from good balance. As stated above, the multi-core computing revolution has begun and will change the landscape of parallel computing. The development

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

of multi-core technology may bring new opportunity to improve performance or solve the workloads balancing problem of parallel discrete event simulation. Our work focuses on the question that whether the overhead of globally scheduling can be cut down by special data structures and algorithms under multi-core environments.

Our work makes the following contributions:

- This work introduces a global event scheduling mechanism based on distributed event queues to reduce synchronization cost, avert workloads imbalance and overcome other problems.
- Based on the global schedule mechanism above, three parallel patterns are presented for simulation model developers to choose from according to the different simulation modeling characteristics.
- The global schedule mechanism and Software Transactional Memory (STM) shared attribute access mechanism are evaluated by thorough experiments.

The rest of the paper is organized as follows. We discuss some related work in the following section. The details of the proposed techniques are presented in Section 3. Experimental evaluations are presented in Section 4, and concluding remarks appear in Section 5.

II. Related Work

The primary objective of PDES for analytic simulation is the as-fast-as-possible execution speed. Two major approaches for addressing the synchronization problem of PDES are conservative [8] and optimistic [6] mechanisms. Conservative mechanisms allow for the simultaneous execution of events only when the system can guarantee that the events are "safe". This protocol works well for simulation models already containing significant parallelism, however the concurrency will not be fully exploited if the lookahead is too small. Optimistic mechanism, which is the focus of this paper, relaxes the strict enforcement on event order. Thus, an optimistically synchronized PDES can sometimes allow causality violations in event executions.

Jefferson's Time Warp [6] mechanism is the most well-known optimistic synchronization algorithm. Since then, various Time Warp systems have been developed and implemented [9-16]. Most of these Tim Warp systems were originally designed for shared memory processing environments [9], [10], [17] or message-based distributed memory platforms [11], [13], [14], [15], [18].

For those that were originally designed for shared memory environments like GTW [9] and ROSS [10], a multi-thread event execution kernel is often used, and the distributed queue local scheduling mechanism is also used in some of the shared memory implementations. The multi-threaded WARPED (namely ThreadedWarped) [19] uses a global priority queue and a manager thread for event synchronizing and scheduling.

"Pull processing" and "push processing" [20] are the mechanisms used by most PDES systems for object state interacting. The drawbacks with pull processing are that two messages are required for each state information collection, and the process requesting the information must usually block until the query has been satisfied. For the push processing, it may require more message transmissions than are really needed, because the source of the data cannot know if the subscriber of this information requires each new value of the state variable. The shared state problem has been well studied in the mid 90's. Ghosh and Fujimoto [21] designed a "space-time memory system" in software that provided a state version control system so that LPs operating under a Time Warp event scheduler on shared-memory systems can easily share state without having to resort to sending messages. In [22], Horst Mehl provided a deep explanation and insight into different variations (six of them) of space-time memory.

III. Methodology

A. Overview

We use a multi-threaded event execution kernel in our implementation, which is the same as many Time Warp systems designed for shared memory environments [9], [10]. The work in this paper only focuses on multi-core environments. How to apply our techniques on the distributed memory/message passing systems will be studied in future work.

We first introduce the partitioning strategy on which our optimal mechanisms are based. Figure 1 shows three parallel simulation partition patterns with different granularities. Figure 1(a) shows a normal pattern, named the LP parallel pattern. Each node stands for a simulation entity/logical process (LP). The event scheduling and rollbacks are carried out on an entity/LP by entity/LP basis.

Figure 1(b) shows a coarse granularity pattern, in which simulation model runs with so-called extended logical processes (Ex-LPs for short) as the parallel units. An Ex-LP is a collection of entities/LPs. Entities/LPs in the same Ex-LP can access state variables of each other directly. This direct access mode can't be traced by later possible rollbacks, so every Ex-LP should manage a processed event-list to hold all processed events of the entities/LPs that belong to this Ex-LP. Rollbacks are done based on an Ex-LP's processed event list. The motivation of this design is to avoid frequent rollbacks that are induced by the frequent interactions among the simulation entities/LPs inside of an Ex-LP. Therefore it needs a certain partitioning algorithm to put entities/LPs that have close causal relationships to each other into one Ex-LP. The goal of this partitioning algorithm is to reduce rollbacks and data communication but still hold certain parallel degree, namely balancing the parallel degree and efficiency. The partition mechanism in our paper only focuses on avoiding frequent rollbacks, while similar partition mechanisms proposed in previous work focus on workload balancing. So unlike common partitioning algorithms, our approach doesn't take into account computational workloads, which will be dynamically balanced by the global schedule mechanism in sub-section 3.2. From the view of the simulation engine code implementation, the "Ex-LP" is very similar to "KP" in ROSS

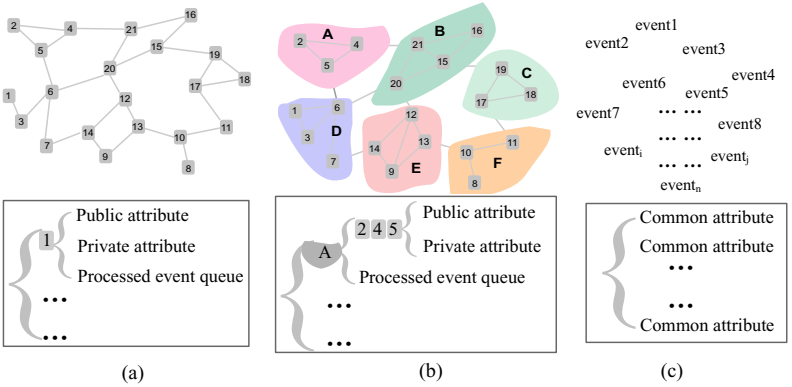


Fig. 1. Parallel patterns with different granularities.

[10], and this concept is proposed here for the modelers' better utilization of our simulation engine.

Figure 1(c) shows a fine granularity pattern, in which a simulation model runs with simulation events as the parallel units. There is no Ex-LP or LP. Thus, there is no need for a processed event-list. This pattern is suitable for the simulation models that can't be easily partitioned into separate entities/LPs or the simulation entities/LPs that have many potential concurrent events inside. In this context, we refer to this pattern as event parallel pattern. For those simulation models which cannot be partitioned into as many LPs as the number of available cores but still have numerous events that can be executed in parallel, the event parallel pattern can be used.

From the view of simulation kernel, the LP parallel pattern can be considered as a special case of Ex-LP parallel pattern (one Ex-LP contains only one LP). So the following discussion we will not discuss the LP parallel pattern. As for the Ex-LP parallel pattern, we distinguish simulation state attributes by classifying them into Ex-LP-private attributes and public attributes.

We define three categories for program variables to convey semantic information about the scope of use, communication, sharing patterns and consistency requirements to the system. The optimistic mechanism could support this classification by providing associated means to conflict detection and versioning for certain regions of memory.

Ex-LP-private attribute: One state variable that can not be accessed by entities in other Ex-LPs is declared as Ex-LP-private attribute. In our scheduling scheme, as will be discussed later, only one event in each EX-LP can be executed in the working thread at one time, so the Ex-LP private variables are not subject to conflicts.

Public attribute: On the contrary, one state variable that can be accessed by entities in other Ex-LPs is declared as public attribute.

Common attribute: As for the event parallel pattern, all the state variables in the simulation model are declared as common attributes.

The optimistic mechanism enables the processing of events

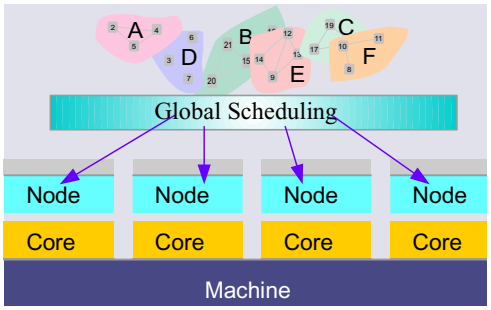


Fig. 2. Global scheduling.

out of their causal order. Causal errors may occur when these events access the public and common attributes. Thus, a concurrency public/common attribute data access control mechanism must be used. The public/common attribute access mechanism will be discussed in Section 3.3. Figure 2 shows the role of global scheduling mechanism for Ex-LP parallel pattern. Ex-LP is not bound to a processing node, but scheduled globally according to the busy or idle states of all the processing nodes. The same approach is used to schedule the event parallel pattern. The following subsection will discuss the implementation of this global scheduling mechanism.

B. Distributed Queue and Global Scheduling (DQ-GS) Mechanism

The PDES program running on multi-core computer has several threads working in parallel. The number of threads typically depends on the processing capabilities (number of cores) of the system that hosts the program. All the simulation events (of different simulation objects) on one machine are scheduled together in timestamp order. This global scheduling scheme is similar to [19], but we don't use a manager thread and global queue of simulation objects. Instead, we create an event list for each thread. Each working thread will repeatedly select the smallest unprocessed event from all the event lists and process it.

During our development of the DQ-GS mechanism, we tested the performance of global scheduling strategy without

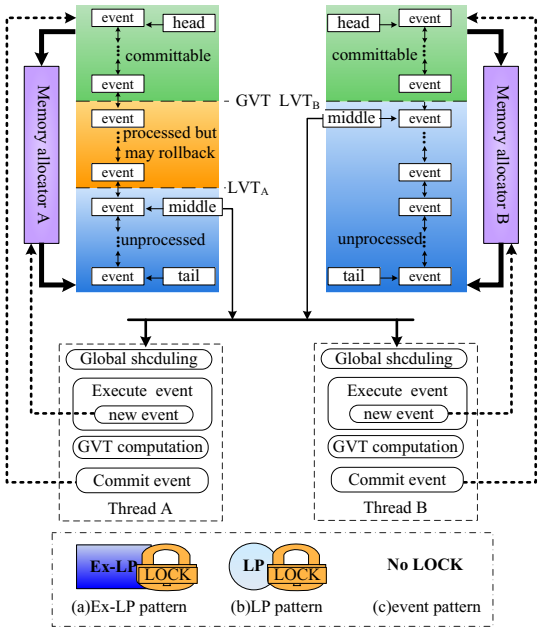


Fig. 3. Data structures and processing flow of DQ-GS mechanism for two thread nodes.

any optimization measures. Although global scheduling can achieve good workloads balance, it takes too much overhead on locking operations (50%∼90%). Therefore, several optimization techniques are applied to decrease this overhead.

Figure 3 illustrates the data structures and process flow of the DQ-GS mechanism. Each thread holds a thread-specific priority queue, referred to as *node_el*, and a thread-local dynamic memory allocator. The processing flow of each thread is the same. There is no service thread or central control thread. Algorithm 1 illustrates the implementation of global scheduling and GVT computation, which are included in each thread's scheduling loop function.

In order to minimize the overhead of locking and cache hit misses, *node_el* is specially designed based on multi-threaded memory management. It holds events which are sorted by timestamps as its elements, and contains the following three pointers: *head* pointer to the smallest time-stamped event, *middle* pointer to the event that will be checked by scheduler in the next loop (i.e., the local smallest time-stamped unprocessed event) and *tail* pointer to the event containing the largest time stamp. So the part between head and middle pointer can be equated with a processed event list, and the remainder part between *middle* and *tail* pointer can be regarded as a future event list.

1) Events Creation and Reclamation: To minimize the locking cost, we designed a mechanism that enables most operations of an event to be done in a lock-free manner. When a running thread is going to generate a new event, it allocates a memory block for the new event from its local storage and this thread is referred to as the event's owning thread. Then, the same thread inserts the new event into its local *node_el*. This event may be executed by other threads, but it

Algorithm 1 Global scheduling and GVT computation of DQ-GS mechanism

LOCK(g_lock)
if $th\_para[my\_id].syn\_flag ==$ TRUE **then**
  $th\_para[my\_id].syn\_flag =$ FALSE
  $syn\_count++$
  $pre\_glbts.reduce\_to(runtime\_LVT)$
  if $syn\_count ==$ NUM_THREADS **then**
    $GVT \leftarrow pre\_glbts$; $syn\_count \leftarrow 0$
    $wait\_syn\_flag =$ FALSE
    for $u = 0$ to NUM_THREADS - 1 **do**
      $th\_para[u].commit\_flag$ = TRUE
    end for
  end if
end if
$gbl\_min\_th\_ts \leftarrow MIN\{min\_th\_ts_1, \ldots, min\_th\_ts_n\}$
$sel\_thread\_ID \leftarrow \{i | min\_th\_ts_i = gbl\_min\_th\_ts\}$
$GVTIntervalCounter++$
if $GVTIntervalCounter \geq GVT_{Interval}$ **and** $wait\_syn\_flag ==$ FALSE **then**
  $wait\_syn\_flag =$ TRUE; $GVTIntervalCounter \leftarrow 0$
  $pre\_glbts \leftarrow gbl\_min\_th\_ts$
  for $u = 0$ to NUM_THREADS - 1 **do**
    $th\_para[u].commit\_flag$ = TRUE
  end for
end if
LOCK($tpq[sel\_thread\_ID].middle\_lock$)
$SelEvent \leftarrow tpq[sel\_thread\_ID].nel.peek\_middle()$
if $SelEvent \neq$ NULL **then**
  $SelEvent \rightarrow nel\_in\_tag$ = FALSE
  $tpq[sel\_thread\_ID].nel.shiftdown\_middle()$
  $min\_th\_ts_{sel\_thread\_ID} \leftarrow tpq[sel\_thread\_ID].nel.get\_mid\_ts()$
  UNLOCK(g_lock)
  UNLOCK($tpq[sel\_thread\_ID].middle\_lock$)
  $runtime\_LVT \leftarrow step\_adv\_opt(SelEvent)$
else
  UNLOCK(g_lock)
  UNLOCK($tpq[sel\_thread\_ID].middle\_lock$)
  $runtime\_LVT \leftarrow$ MAX_TIME
end if

can not be deleted from its owning *node_el* by other threads. A canceled event cannot be immediately deleted but has its *invalid* flag set TRUE. The deletion and memory reclamation of an event are performed only by its owning thread after each GVT computation. Only in the case that the new event's time stamp is smaller than the time stamp of the event referenced by *middle* pointer needs a locking operation on modifying the *middle* pointer.

2) Global Scheduling: In each scheduling loop of a thread node, it selects the global smallest time-stamped event from the set of thread-local smallest time-stamped unprocessed events. The selected event will not be deleted from its owning *node_el*, just as its *nel_in_tag* flag set FALSE and shifts the

middle pointer to the next unprocessed event. The *middle* pointer is used to avoid the delete operation of a scheduled event from the future event list and the insert operation of a processed event into the processed event list. In this way, the critical global lock for global scheduling is held for only a few instructions.

3) Event Execution: As for the *Ex-LP/LP* parallel pattern, only one event for each Ex-LP/LP is executed at any time. This avoids runtime conflict detection on *Ex-LP/LP*-private variables. However, state saving still should be done on *Ex-LP/LP*-private variables. A Boolean flag named *busy* of each *Ex-LP/LP* guarantees that only one event of this *Ex-LP/LP* can be executed in the working thread. We note, each simulation event maintains a pointer to its owner *Ex-LP/LP*.

When an event is selected for processing by a thread, as done for the event parallel pattern, the thread executes the selected event directly, as for the *Ex-LP/LP* parallel pattern, the thread calls the processing method of the event's owning *Ex-LP* or *LP*. The processing method first checks whether the *Ex-LP*'s or *LP*'s *busy* flag is set. If the *busy* flag is set, which indicates that another thread is executing its events, the proper processing procedure puts the current selected event into the *Ex-LP*'s *ready_el* (ready event list) and then returns. If the *busy* flag is not set, the thread executes the event. Whenever an *Ex-LP* finishes executing an event, it attempts to access its *ready_el* and executes all the events in the *ready_el*. Once there is no event in the *ready_el* to execute, the processing procedure unsets the *busy* flag and then returns.

4) Rollback Scheme: The processed event list as previously discussed in this section is referred to as *pel* in our implementation. In order to reduce memory allocation and reclamation operations, each event only uses one block of memory on the local machine. Two pairs of pointers are stored in an event's data structure to link the event into the thread-specific *node_el* and the *pel* list respectively. As previously discussed in this section, the deletion and memory reclamation of an event are performed by its owning thread. In order to avoid frequently locking the *pel* which includes the ready-to-delete event, the event's owning thread won't access the *pel* during the deletion phase. But segmentation faults may occur because a freed event's memory location (done by the event's owning thread) may be referenced during rollback phase of the events in *pel*. We solve this problem by introducing a field named *pprev_ts* in each linked event. The *pprev_ts* variable records the timestamp of its previous event in *pel*. Operations on *pel* always begin with the tail of *pel* and access one by one. Prior to shifting the access pointer to the previous event, it must check the *pprev_ts* field of current access location to make sure it is greater than GVT. Since only the events with timestamps that are less than GVT can be freed, segmentation fault won't occur.

5) Asynchronous GVT Computation: DG-TW uses an asynchronous GVT computation algorithm, which is included in Algorithm 1. Because events in a thread's *node_el* may be executed by other threads and the new generated events are inserted into other thread's *node_el* which may potentially hold an earlier timestamp, the local time of a thread is now not as simple as grabbing the minimum timestamp event from the unprocessed events queue. To track the local time, each thread maintains a local variable called *runtime_LVT* that contains the minimum timestamp of any event generated during each scheduling loop. All inter-thread communication is realized by using a group of global variables as described below. The variable *th_para* is an array which holds each thread's status flags. The *GVTinterval* is a parameter that controls the frequency with which GVT is calculated, just like [9] and [10].

When a thread detects that the current value of *GVTIntervalCounter* is over *GVTinterval*, the thread will initiate a new GVT computation by setting all the thread's *syn_flag* in *th_para*, which indicates whether the thread should report local *runtime_LVT*. Then, combined with event scheduling, the time stamp of the selected smallest time-stamped event to schedule is assigned to *pre_glbts*, which is a lower bound on the time stamps of any unprocessed events in all the *node_el*s at the moment GVT computation is initiated.

Each thread checks its *syn_flag* at the beginning of the main event-processing loop and notes whether the flag was set. If *syn_flag* has already been set at the beginning of the loop, the thread reports its local *runtime_LVT* by reducing the *pre_glbts* to its *runtime_LVT* (i.e., if *runtime_LVT* is less than *pre_glbts*, then the value of *runtime_LVT* is written into *pre_glbts*). It then increases *syn_count* to indicate that it has reported its local *runtime_LVT*.

The last thread who reports its local *runtime_LVT* (the thread that increases *syn_count* to the number of threads in the system) assigns the current *pre_glbts* to GVT value and sets all the thread's *commit_flag* in the *th_para* array, which triggers the routine for committing processed events.

A lock on the above operations ensures that at most one thread can modify the global variables.

C. Shared Attribute Access Mechanism Based on Transactional Memory

As described earlier, a mechanism must be used to ensure that shared public/common attribute accesses do not violate causality. This subsection uses a solution that can fully take advantage of the shared memory address space that exists between multi-threads on a multi-core machine.

Our solution uses software transactional memory (STM) mechanisms that automatically coordinate shared attributes reading and writing between threads with simple program interfaces. The key idea is to take simulation events as ordered transactions. Simulation events in the same process access shared attributes directly in the common memory address space. There are many available proposals that implement TM in software, hardware or a hybrid of the two [24-27]. But the current transactional memory implementations do not work very well with PDES. The reason is that simulation events in PDES system often schedule new events with future time stamps, but current TM systems only detect conflicts in a fixed group of transactions. So the TM for Time Warp algorithm, referred to as TW-TM, should be specially designed.

We designed a TW-TM system in software, TW-STM for short, according to the features of Time Warp algorithm. The data structures, programming model, synchronization mechanism and conflict solving scheme of our TW-STM system is similar to the "space-time memory system" in [21], [22]. The differences are the details for public attributes and memory reclamation scheme.

As described in section 3.1, we divide shared attributes into public attributes and common attributes. The records of history information on accessing to common attributes is the same as the previous related work. As for the public attributes, the *ReadList* of each data version records all the exterior events (as for Ex-LP or LP parallel pattern, the events of other Ex-LP or LP) that have read the value of this data version but no need for interior events . An event records all the data versions that it has written to exterior public/common attributes in its *WriteList* but no need for its own Ex-LP/LP's public attributes.

In our scheme, each shared attribute object has its specific memory management. The allocation and reclamation of memory for data versions and read descriptors are performed by the specific memory managers of their owning shared attributes.

In previous related work, fossil collection is initiated to reclaim storage from invalid versions and versions with version numbers less than the global virtual time of the simulation. A read operation to a shared object returns a pointer to that version of the object which has the highest timestamp less than that of the event that invoked the ReadObj primitive and if no such version exists, an error status is returned. But in our scheme, having an update time stamp less than GVT is not sufficient for a data version to be reclaimed. Because if it has been freed and there is no safe data version after it, later events may access a null (unavailable) data. So a data version that can be reclaimed must met the requirement that the update time-stamp of the next data version (referenced by *next* pointer) in the *dv_list* is less than GVT. For example, the item $Item_0$ of $\mathbf{shared-attri_2}$ in Figure 4 is eligible for deletion, but the item $Item_0$ of $\mathbf{shared-attri_1}$ is ineligible for deletion. Furthermore, when creating a public/common attribute an initialed data version value must be given. So it won't happen that no appropriate version exists for a read operation.

In order to reduce the overhead for reclamation, both *WriteList* and *ReadList* are implemented as a linked list. Elements in *ReadList* link each other by *d_prev* pointer in read descriptor and *WriteList* by *e_prev* pointer in data version. Shared attribute's write or read method is responsible for reclaiming their memory (batch or on-the-fly fossil collection strategy [20]). Event object only holds the head pointer of its *WriteList*. So, an event's commit function needn't go through its *WriteList*.

IV. Experiment and Evaluation

To fully evaluate the performance distinction between different mechanisms, three tiny Time Warp kernels are implemented using different mechanisms: our multi-threaded distributed queue global schedule mechanism (DQ-GS-MTH for short), the multi-threaded distributed queue local schedule mechanism (DQ-LS-MTH for short) and a multi-process version of distributed queue local schedule mechanism (DQ-LS-MP for short). All these tiny kernels are implemented by C++. The DQ-LS-MP uses the shared memory version of libSynk [11] for synchronization and time management.

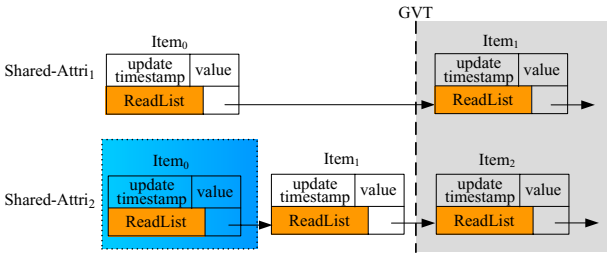


Fig. 4. Requirement for fossil collection in TW-STM.

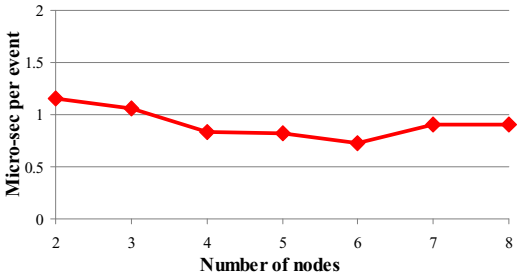


Fig. 5. Performance of DQ-GS mechanism on the changing of processing nodes.

All tests would run on a single computing node of a Linux Cluster. The node is equipped with two way 2.53GHz QuadCore Xeon Processors E5540, 8GB RAM. The operating system is Red Hat Enterprise Linux 5.0. The GCC version is 4.1. PHOLD [28] is used as the evaluation benchmark.

In our PHOLD implementation, *NLP* simulation logical processes are globally scheduled by all available threads. Each logical process generates one event at initialization. When a logical process receives an event, it schedules a new event into the future with a time increment computed by the following equation: $lookahead + EXP(1)$, where $lookahead$ is a minimum time increment; $EXP(1)$ stands for the exponential distribution with a mean value of 1.0. We use a uniform random number generator (URNG) to randomly determine event destinations, and another URNG stream to determine time increment. So fixed costs of an event include runtime overheads of the simulation engine as well as random number generation costs. In some tests, we put an extra *workload* (a group of float instructions) on each event to reveal performance for different event granularity.

A. DQ-GS Mechanism Performance

1) Test1: Performance Measurement - Number of Processing Nodes: In this test, the events are configured with uniform distributed random destinations; $lookahead$ is set to 0; there is no extra workload for any event. Figure 5 shows the

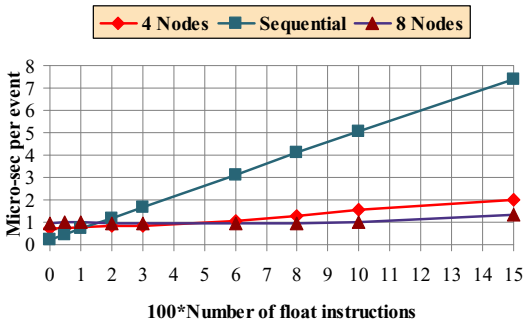


Fig. 6. Performance variation on the changing of workloads.

average processing time per event while the number of threads (processing nodes) is varied. The experiment result reveals DQ-GS mechanism scales excellently with the number of threads.

2) Test2: Performance Measurement - Workloads of Events: Figure 6 shows the average time taken to process an event in PHOLD, for increasing the number of threads (processing nodes) and workloads. Other configurations are the same as *Test1*. As the *workload* configurations grow to 120 float instructions, the DQ-GS-MTH with 4 processing nodes begins to outperform sequential execution. As the *workload* configurations grow to 150 float instructions, the DQ-GS-MTH with 8 processing nodes begins to outperform sequential execution. As the *workload* configurations grow to 520 float instructions, the DQ-GS-MTH with 8 processing nodes begins to outperform 4 processing nodes. The speedup of parallel execution to sequential execution is more and more close to the number of processing nodes as the workloads increase.

3) Test3: Performance Measurement - lookahead and the Density of Events: In the test of Figure 7(a) and Figure 7(b), the number of processing nodes (threads for DQ-GS-MTH and DQ-LS-MTH, processes for DQ-LS-MP) is set to 8. The value of lookahead grows with the x-axis, and other configurations are the same as *Test1*. In the test of Figure 8, the density of events grows with the x-axis, and other configurations are the same as *Test1*. As the parent event only generates one new child event in our PHOLD benchmark, the density of events is defined as $\frac{NLP \times R}{N_{ProcessingNodes}}$, where R is the ratio of the number of events to the number of processes. Because in our test each logical process generates one event at initialization, R is equal to 1 here. In the test of Figure 8, the value of *lookahead* is set to 1. The workloads of events grow with the x-axis, and other configurations are the same as *Test1*.

Figures 7 and Figure 8 show that the DQ-GS-MTH can greatly reduce the rollback rates and event processing cost. Figure 8 shows that as the workloads of each event grow, the DQ-GS-MTH outperforms DQ-LS-MTH more and more. The reason for this is attributed to lower rollback rate.

4) Test4: Performance Measurement - the Locality of Events: If the destination of a new event is the current parallel processing node, then this event is called a "local" scheduled event (local event for short). Otherwise, this event

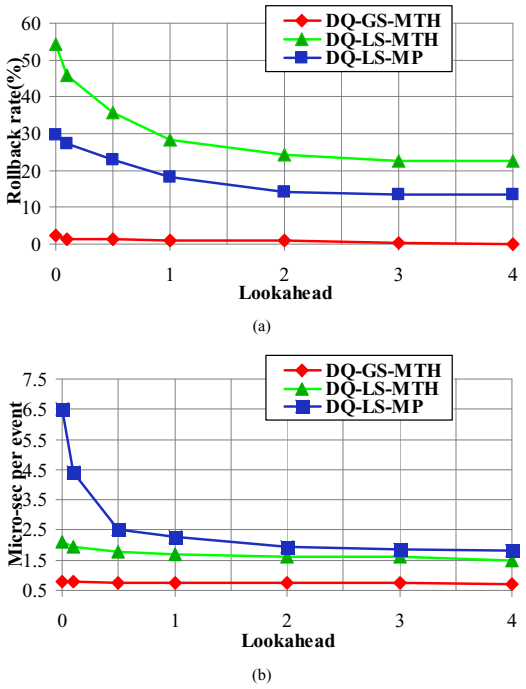


Fig. 7. Rollback rates & performance variations on the changing of *lookahead*.

is called a "remote" scheduled event (remote event for short). The "locality" of events is defined as the total number of local events divided by the total number of scheduled events during simulation. It is obvious that the sum of "locality" and percentage of remote events should be 100%.

In the test of Figures 9, the number of processing nodes is set to 8, the "locality" of events grows with the x-axis. The workloads of each event are set to 0, and other configurations are the same as *Test1*. In the test of Figure 10, the "locality" of events is set to 95%, and the workload of each event grows with the x-axis.

Figure 9(a) shows that the rollback rate of DQ-LS-MTH is higher than DQ-LS-MP. As the "locality" grows over to 80%, the rollback rate of DQ-LS-MTH becomes even higher until the "locality" reaches 100% (this is also observed in ROSS which is implemented with C and famous for its higher event rate [9]), which is very different to the multi-process execution. Investigation into this phenomenon reveals that the higher rollback rate is a side effect of the higher event rate of the multi-threaded local-schedule mechanism compared to the multi-process execution. In contrast with this, DQ-GS-MTH demonstrates stable, highly efficient execution. Figure 9(b) shows that DQ-GS-MTH outperforms DQ-LS-MTH in the whole test, and the DQ-GS-MTH outperforms DQ-LS-MP when the "locality" is below 95%. Locality with 95% may be a worse case of DQ-GS-MTH, but in the test of Figure 10, when the "locality" of events is set to 95%, certain speedup still be achieved as the workloads of each event grow with the x-axis.

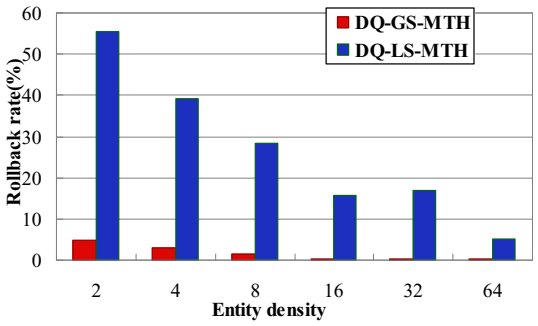


Fig. 8. Rollback rates variations on the changing of the density of events.

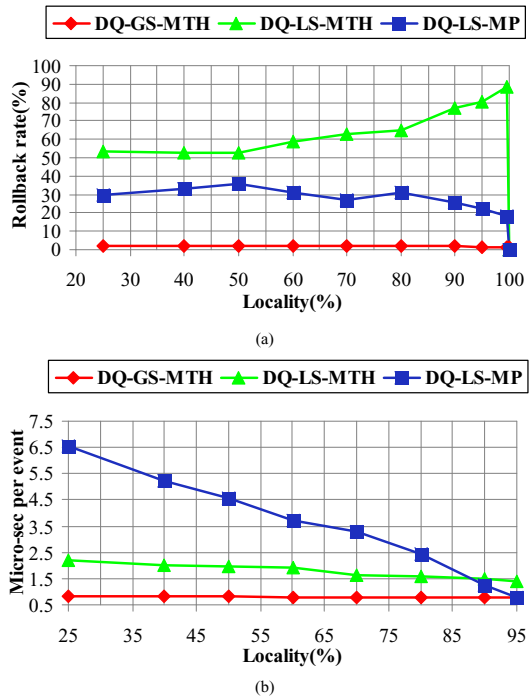


Fig. 9. Rollback rates & performance variations on the changing of "locality".

B. TW-STM Mechanism Performance

To evaluation our TW-STM mechanism, both the "pull" mechanism and our TW-STM are implemented on the basis of DQ-GS-MTH. In the test of Figures 11 and 12, the number of processing nodes is set to 6, the workloads are set to 0, and both the destination of a "pull" event and the visited ID of a public attribute in TW-STM are randomly generated. In Figure 11, both the number of logical processes and the number of public shared attributes are three times of processing nodes. The x-axis represents the attribute/state query probability of an event. The Figure 12 shows the performance and rollback rate variations on the changing of the number of shared attributes (the attribute query probabilities are set to 40%). The x-axis is the total number of Shared-Attributes and all entities are sharing this amount of attributes. In fact, the competition for shared attributes is lower when there are more shared attributes. This test is a performance measurement of high

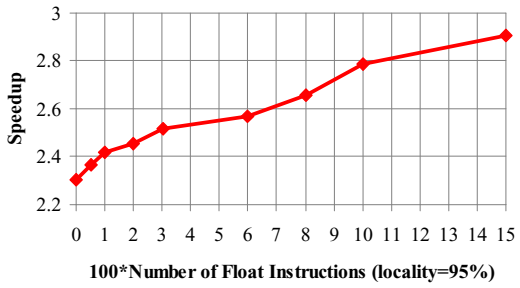


Fig. 10. DQ-GS-MTH speedup variations on the changing of the workloads (speedup relative to DQ-LS-MTH).

competition for shared attributes. Figures 11 and 12 show that TW-STM outperforms "pull" mechanism when attribute/state query competition rate is high.

V. Final Remarks and Future Work

In this paper, we proposed two mechanisms to improve the performance of Time Warp system on multi-core environments. Compared with the conventional distributed event queue and local schedule mechanism, the experiment results showed that the global schedule mechanism proposed in this paper can effectively decrease rollback rate and balance the workloads at a low event scheduling cost. The TM-based object shared attribute/state access mechanism is easy to be used and outperforms the conventional "pull" mechanism.

For future work, we plan to further improve our Time Warp kernel by using look-free algorithm and other techniques that can decrease the cache miss rate. Furthermore, we propose to find an alternative to the coarsely granular locks that are required to mutex accesses to the shared attributes. We are working on applying our proposed mechanisms on some real simulation applications to evaluate the performance.

Acknowledgments

Thanks to the anonymous referees who provided excellent feedback on this work, and special thanks to professor Carothers for his great help on our paper. This work is partly supported by the National Science Foundation of China (NSF) (General Grant No. 60773019 and No. 60903223), and the Ph.D. Programs Foundation of Ministry of Education of China (Grant No. 200899980004). The authors would also like to thank the open-source platforms ROSS and MUSIK.

References

[1] http://x10.codehaus.org/
[2] http://www.threadingbuildingblocks.org/
[3] http://software.intel.com/en-us/articles/intel-mkl/
[4] G. S. Sohi, S. E. Breach, and T. N. Vijaykumar. Multiscalar processors, in the 22th ISCA, June 1995.
[5] M. Herlihy and J.E.B Moss, "Transactional memory: architectural support for lock-free data structures," in the 20th ISCA, New York, NY USA: ACM, pp. 289-300, May 1993.
[6] D. R. Jefferson, "Virtual time," ACM Transactions on Programming Languages and Systems, vol. 7, no. 3, pp. 404-425, 1985.
[7] Sina Meraji, Wei Zhang, and Carl Tropper, "A Multi-State Q-learning Approach for the Dynamic Load Balancing of Time Warp," in Proceedings of the 24th Workshop on Parallel and Distributed Simulation (PADS'10), 2010.

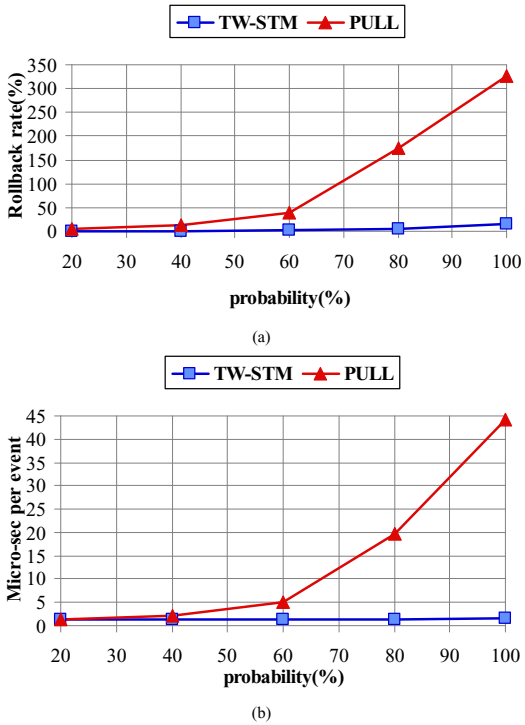


Fig. 11. Rollback rates & performance variations on the changing of state query probability.

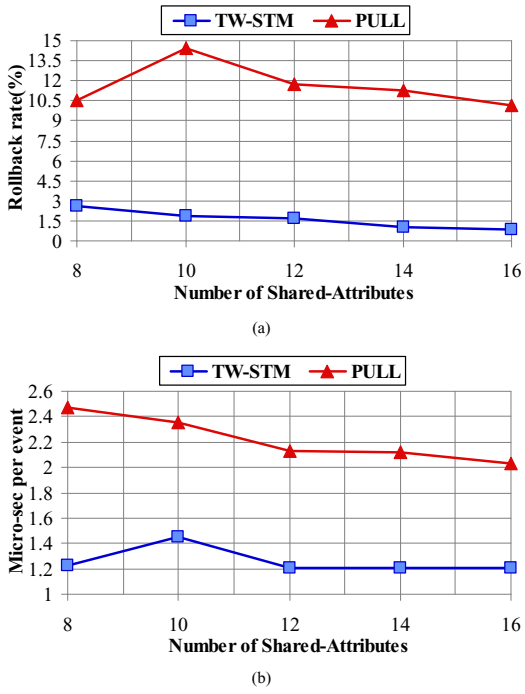


Fig. 12. Rollback rates & performance variations on the changing of the number of shared attribute.

[8] R. E. Bryant, "Simulation on a distributed system," in Proceedings of the 1st International Conference on Distributed Computing Systems, pp. 544-552, 1979.

[9] S. Das, R. Fujimoto, K. Panesar, D. Allison, and M. Hybinette, "Gtw: a time warp system for shared memory multiprocessors," in the 26th WSC, (San Diego, CA, USA), pp. 1332-1339, 1994.

[10] C. D. Carothers, D. Bauer, and S. Pearce, "Ross: A high-performance, low memory, modular time warp system," in Journal of Parallel and Distributed Computing, pp. 53-60, 2000.

[11] K. S. Perumalla, "musik: A Micro-kernel for Parallel/Distributed Simulation Systems," in Proceeding of the Workshop on Parallel and Distributed Simulation, pp. 59-68, 2005.

[12] R. Bagrodia, M. Takai, Y. an Chen, X. Zeng, and J. Martin, "Parsec: A parallel simulation environment for complex systems," IEEE Computer, vol. 31, pp. 77-85, 1998.

[13] J. Steinman, "Speedes: A unified approach to parallel simulation," in Proceeding of the 6th Workshop on Parallel and Distributed Simulation, pp. 75-83, 1992.

[14] Jeff. Steinman, "Introduction to Parallel and Distributed Force Modeling and Simulation," in Proceeding of the Spring 2009 Simulation Interoperability Workshop, 09S-SIW-041, 2009.

[15] H. Avril and C. Tropper, "Clustered time warp and logic simulation," in the 9th PADS, pp. 112-119, 1995.

[16] D. E. Martin, T. J. McBrayer, and P. A. Wilsey, "Warped: A time warp simulation kernel for analysis and application development," in HICSS '96: Proceedings of the 29th Hawaii International Conference on System Sciences Volume 1: Software Technology and Architecture, (Washington, DC, USA), p. 383, IEEE Computer Society, 1996.

[17] J. Liu, D. M. Nicol, and K. Tan, "Lock-free scheduling of logical processes in parallel simulation," in the 15th PADS, pp. 22-31, May 2001.

[18] R. L. Bagrodia, K. M. Chandy, and J. Misra, "A message-based approach to discrete-event simulation," IEEE Trans. Softw. Eng., vol. 13, no. 6, pp. 654-665, 1987.

[19] Ryan James Miller. Optimistic Parallel Discrete Event Simulation on a Beowulf Cluster of Multi-core Machines. Master Dissertation, Cicinati University, July, 2010.

[20] Richard M. Fujimoto. Parallel and Distributed Simulation Systems, JOHN WILEY&SONS, INC, 2000.

[21] K. Ghost and R. M. Fujimoto, "Parallel Discrete Event Simulation Using Space-Time Memory," in 20th International Conference on Parallel Processing (ICPP), August 1991.

[22] Mehl, H. and Hammers, S., "How to Integrate Shared Variables in Distributed Simulation," Technical Report SFB124-32/92. Dept. of Computer Science, University of Kaiserslautern, Germany, 1992.

[23] Bauer Jr., D. W., C. D. Carothers, and A. Holder, "Scalable time warp on blue gene supercomputers," In the 23rd PADS, 35-44. Washington, DC, USA: IEEE, 2009.

[24] Chi Cao Minh, Martin Trautmann, JaeWoong Chung, Austen McDonald, Nathan Bronson, Jared Casper, Christos Kozyrakis, Kunle Olukotun, "An Effective Hybrid Transactional Memory System with Strong Isolation Guarantees," ISCA'07, June 9-13, 2007, San Diego, California, USA.

[25] P. Felber, C. Fetzer, and T. Riegel, "Dynamic Performance Tuning of Word-Based Software Transactional Memory," in the 13th PPoPP, 2008.

[26] M. Herlihy and E. Koskinen, "Transactional boosting: methodology for highly-concurrent transactional objects," in PPoPP '08: Proceedings of the 13th ACM IGPLAN Symposium on Principles and practice of parallel programming, pages 207-216, New York, NY, SA, 2008. ACM.

[27] M. F. Spear, V. J. Marathe, W. N. S. III, and M. L. Scott, "Conflict Detection and Validation Strategies for Software Transactional Memory," in 20th Intl. Symp. on Distributed Computing (DISC), 2006.

[28] R. M. Fujimoto, "Performance of Time Warp Under Synthetic Workloads," in Proceedings of the SCS Multiconference on Distributed Simulation, vol. 22, SCS Simulation Series, 1990, pp. 23-28.

On the Accuracy of Ad Hoc Distributed Simulations for Open Queueing Network

Ya-Lin Huang, Christos Alexopoulos, Richard Fujimoto, and Michael Hunter
Georgia Institute of Technology
Atlanta, GA 30332, U.S.A.
huangyl@cc.gatech.edu, christos@isye.gatech.edu, fujimoto@cc.gatech.edu, michael.hunter@ce.gatech.edu

***Abstract*—Ad hoc distributed simulations are constructed by embedding online simulations into a network of sensors. Recent work examining the effectiveness of ad hoc distributed simulations in modeling open queueing networks has been limited to a specific network configuration. This paper examines the accuracy of the ad hoc approach more broadly for open queueing networks. First, we identify conditions where the ad hoc approach will yield accuracy comparable to traditional replicated simulations. For cases where these conditions do not apply, we propose a "buffered-area" mechanism to improve the accuracy of system performance estimates. Empirical evidence is presented showing that this mechanism helps achieve a substantial bias reduction with a moderate increase in execution time.**

I. INTRODUCTION

Ad hoc distributed simulations [1] are online simulations embedded in a sensor network to monitor, analyze, and/or optimize a system of interest. Traditionally online simulation of an operational system is performed in a centralized manner; data from the sensors are streamed to a central location where the simulation is performed. By contrast, the ad hoc approach embeds the simulation computation into a network of sensors. Each sensor autonomously performs an online simulation of some portion of the system. An ad hoc distributed simulation utilizes communication and synchronization services to model the entire region covered by the sensor network.

An ad hoc distributed simulation involves a set of autonomous simulators (or logical processes), each modeling a portion of the system under investigation. Each logical process (LP) autonomously selects the region it models; it might simply model the region covered by its local sensor. Several LPs may model the same physical area, resulting in multiple estimates for that portion of the system. Other parts of the system may not be covered at all. Predictions of future states exchanged among LPs may later be proved inaccurate, e.g., due to unexpected changes in the monitored system. The predictions are recomputed via a rollback mechanism not unlike that used in Time Warp [2]. Details of this mechanism are in [1, 3].

Recent work studied the effectiveness of ad hoc distributed simulations on an 8×8 grid-like open queueing network [4]. Multiple network partitioning methodologies were studied where a renewal model was used to approximate the arrival processes of units entering subnetworks from internal nodes. While the accuracy of the steady-state server utilization estimates seems in general competitive to that of the corresponding sequential simulations, estimation bias is apparent in the steady-state mean queue-length estimates.

In this paper we first show that under certain restrictive conditions the ad hoc approach for open queueing networks will produce output statistics comparable to sequential simulations of the entire system under investigation. For the remaining cases, we propose a buffered-area mechanism to mitigate the possible estimation bias. This mechanism builds upon the idea of enlarging the "distance" between the nodes where the imperfect input approximations are made and where the system performance estimates are measured.

II. RELATED WORK

As mentioned above, [4, 6] illustrated that the validity of the steady-state mean queue-length estimates relies heavily on how well one can model the unit flows crossing the subnetworks modeled by different LPs. Based on the methodology developed by Whitt [5], these flows were modeled by renewal processes. The mean queue-length estimates at heavily-loaded nodes with highly-variable service times were observed to exhibit relatively high bias. The correlation structure of departure processes from queueing systems has been studied quite extensively [7–10] and several approximation methods have been proposed [5, 11–13].

This stochastic process approximation issue can be viewed as an input uncertainty problem. The inputs to each individual LP are arrival processes. In particular, the arrival processes from internal nodes are hard to estimate because (1) there may be insufficient observations, and (2) the complexity of the autocorrelation structure. Several input uncertainty studies aim at adjusting the confidence intervals (CIs) for the point estimates of system performance measures. The risk of ignoring such an issue is discussed in [14]. Solutions include the Bayesian method [15–17], the bootstrap method [18], and the interval-based method [19]. Compared to our proposed mechanism, these approaches require substantial computation time making them poorly suited for the dynamic, real-time environment targeted by ad hoc distributed simulations.

III. ACCURACY OF AD HOC QUEUEING SIMULATIONS

In this section we consider sufficient conditions for ad hoc distributed simulations of a queueing network to produce

This work was supported by the NSF EFRI ARES-CI Grant 0735991.

asymptotic results that are statistically equivalent to those produced by traditional, sequential simulations of the entire network. We focus on the accuracy of point estimates and CI coverage of the system performance measures at each queueing station.

We rely on the partitioning methodology where each LP models a single subnetwork. The internal arrivals for a subnetwork are the departures from the queueing stations surrounding the subnetwork. We refer the links along which these arrivals enter the subnetwork as internal input links. The models that capture arrival processes on input links are important for the accuracy of the output measures produced by the simulations of the subnetwork. Typically, one cannot know a priori the true arrival processes on input links; approximations must be made. We are concerned with steady-state performance measures and assume that the queueing network under study is stable.

Proposition 1. Suppose that (1) the subnetworks modeled by the LPs form a partition of the entire network; (2) the steady-state flow on every internal input link forms a renewal process, and the interarrival times are used to build and update empirical distributions; and (3) the arrivals on internal input links are generated following the periodically updated empirical distributions. Then, as the simulation run length increases, the ad hoc approach yields point estimators and CIs for steady-state performance measures at queueing stations that are statistically equivalent to those produced by a sufficiently long execution of sequential simulations.

Proof: Recall that the external arrivals feeding the network are generated in both cases from the appropriate theoretical models. After a sufficiently long transient phase, with probability 1, the empirical interarrival-time distributions across the internal input links will converge weakly to the respective theoretical distributions [20, Section 20]. Since the interior of each subnetwork is simulated properly, under some mild continuity assumptions, the point estimates for mean performance measures will converge to the respective steady-state means [21]. Under appropriate mixing or ergodicity conditions [22], one can also establish the asymptotic normality of the point estimates and the equality of the underlying variance estimate with that from sequential simulations. ■

The aforementioned assumptions are clearly onerous. In general, the stationary input and output processes at queueing stations in the networks with complex routes are dependent processes with complex multivariate distributions [7–9, 13]. As Table I in [23] shows, very few queueing stations have renewal output processes. Fitting multivariate models to such processes and generating realizations from the fitted models is a hard problem with computational requirements that can overwhelm those imposed by the synchronization mechanism in ad hoc queueing simulations.

Based on Table I, two types of networks meeting the above restrictions are acyclic Jackson networks, and tree-like networks with GI/D/*m* queueing stations and state-independent unidirectional routing. In the first case, the steady-state input processes at queueing stations are superposition of Poisson processes since the routing mechanism at each queueing station forms independent output Poisson processes. In the second case, the splitting mechanism at each queueing station results in delayed (albeit dependent) renewal flows to the emanating links. In fact, the acyclic nature of both classes of systems allows the establishment of the asymptotic validity of estimators for additional performance measures, such as mean travel times of units across paths.

TABLE I. QUEUEING STATIONS PRODUCING RENEWAL OUTPUT PROCESSES IN STEADY STATE

Queueing Station	Output Process	Description
M/M/*m*	Poisson	Poisson arrivals, exponential service times, and *m* identical servers
M/0/*m*/*L*	Poisson	Service times all 0; *L* being system capacity
M/GI/1/1	Renewal	General independent and identically distributed service times with no waiting capacity (excluding the one in service)
M/D/1/1	Renewal	Constant service times
M/GI/∞	Poisson	Infinite number of servers
M/GI/*m* with PS service discipline	Poisson	Process-sharing (PS) service discipline
M/GI/*m*/*L* with LCFS-PR service discipline	Poisson	Last-come-first-serve (LCFS) preemptive-resume (PR) service discipline
GI/D/*m*	Renewal	Renewal arrivals
Note: All systems have infinite system capacity, infinite calling population, and first-come-first-serve service discipline, unless otherwise specified.		

Further extensions are challenging; for example, superposition of independent renewal processes is not a renewal process. The buffered-area mechanism in Section 4 aims at reducing the estimation bias due to the renewal approximations by "filtering" units prior to reaching boundary nodes. Our experiments indicate that the ad hoc approach appears to work well for general queueing networks.

IV. BUFFERED-AREA MECHANISM: EXPERIMENTS AND RESULTS

The buffered-area mechanism aims at improving the accuracy of the performance measures by incorporating an extra portion at the border of each subnetwork modeled by LPs. The enhancement targets at reducing the impact from the approximations made at the input links. The fundamental idea behind this mechanism is to "link" each subnetwork with its neighbors. For example, consider the bidirectional tandem queueing network in Fig. 1. Each node represents a queueing station and the served units flow via the links connecting the nodes. Suppose that an LP is interested solely in the performance measures of nodes G, H, and I. Instead of modeling only these three nodes, the LP enlarges this subnetwork by adding the nodes F and J so that the arrival process approximation is made at the input links to nodes F and J (the arrows with filled head). The LP shares the statistics of the departure processes on the output links of nodes G, H, and I (the arrows with empty head), but not those regarding the departure processes at nodes F and J.

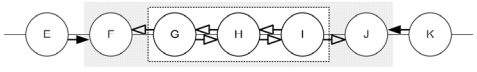


Figure 1. Bidirectional Tandem Queueing Network

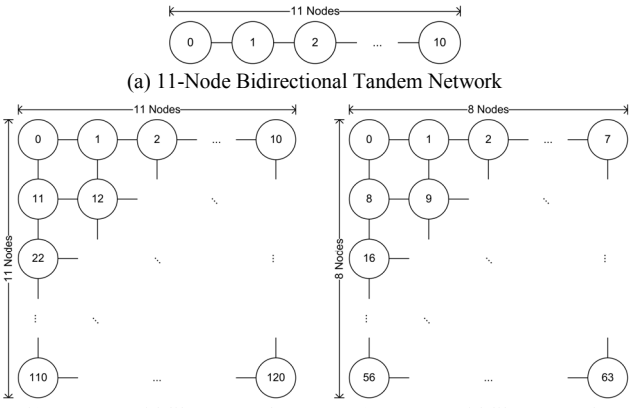
The effectiveness of the buffered-area mechanism depends on the extent to which the approximation bias can be "washed away" by increasing the distance between where the output measures are taken and where the input approximations are made. The longer the distance, the larger the buffered area, which in turn implies higher simulation cost.

The output accuracy gain depends on several factors, including the network topology, connectivity, and the routing mechanisms. We study the output accuracy gain with respect to sizes of the buffered areas on three queueing networks: an 11-node bidirectional tandem network, an 11×11 grid-like network, and an 8×8 grid-like network (Fig. 2). The last study extends an example in [4] to illustrate that the buffered-area mechanism lessens the overestimation issue raised there.

The fundamental mechanisms of the ad hoc queueing simulations are as follows. The LPs update the statistics of the interdeparture times every 30 seconds via a space-time memory [24, 25]. At the start of the simulation, the arrivals at the input links are modeled as Poisson processes with rate λ. Thereafter the arrival processes are assumed to be renewal processes with gamma interarrival times. The scale and shape parameters of the gamma distribution (denoted by α and β, respectively) are dynamically estimated using the data from the rollbacks. The rollback detection function is based on an acceptable range, which is constructed following a quality control paradigm. Since there may be several predictions from the LPs modeling the same area, the data retuned to the LPs are generated using a kernel density estimation approach. Additional details are given in [4, Section 4].

A. Case 1: 11-Node Bidirectional Tandem Queueing Network

This case focuses on the queueing network with 11 nodes in tandem (Fig. 2 (a)). Each node is a single-server, infinite-capacity queueing station where the external arrivals form a Poisson process with rate λ per second; the service times are independent and identically distributed (IID) from the gamma distribution with mean equal to 1 second (denoted by gamma (α, β) with $\alpha\beta = 1$). The served units at the internal nodes move to the neighboring nodes with equal probability p or leave the network (with probability $1 - 2p$). Those at the border nodes (i.e., nodes 0 and 10) continue to a neighboring node with probability p or leave the network (with probability $1 - p$).



(a) 11-Node Bidirectional Tandem Network

(b) 11×11 Grid-like Network (c) 8×8 Grid-like Network

Figure 2. Queueing Networks

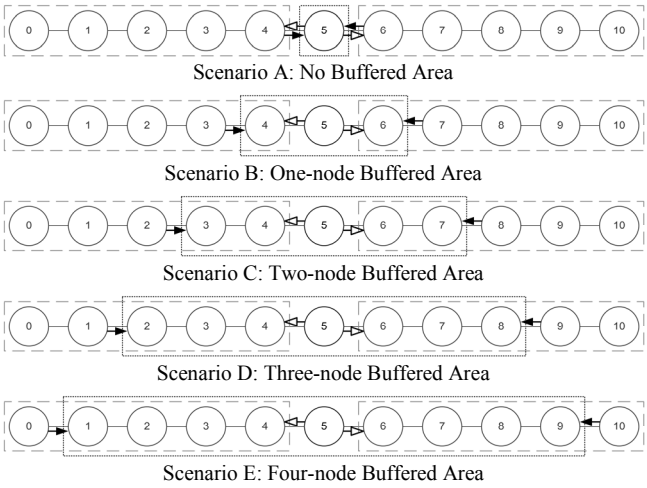


Figure 3. Application of Buffered-area Mechanism to 11-Node Bidirectional Tandem Queueing Network

To simplify the analysis, the entire queueing network is partitioned into three portions: the first consists of nodes 0–4, the second consists of node 5, and the third consists of nodes 6–10. Only the LPs modeling the portion with node 5 adopt the buffered-area mechanism. The sizes of the buffered area differ across the scenarios; see Fig. 3. These LPs update the statistics of the departure processes on the output links of node 5 (the arrows with empty head in Fig. 3) and request arrival information from different input links across the scenarios (the arrows with filled head in Fig. 3).

The evaluation is based on the estimation of the steady-state mean queue length of node 5. Three sets of experiments are designed. The first two involve a heavily-loaded network with traffic intensity about 0.81 at node 5; these studies show empirically that the buffered-area mechanism mitigates the severe bias issues (Sections 4.1.1 and 4.1.2). The third experiment (Section 4.1.3) concerns a network with low traffic intensity at node 5 (about 0.28); in this case we intend to show that the buffered-area mechanism does not deteriorate the output accuracy.

To alleviate the effects of the initial transient, all experiments start collecting data pertinent to the queue-length estimates after 3 hours of simulation time and last for 10 hours. Ten independent runs are performed. Within every run, each portion is modeled by 10 LPs. The results are compared against those from 100 independent sequential runs to equate the number of replications allocated to node 5.

1) Case 1 (a)

This heavily-loaded queueing network has external Poisson arrivals with rate $\lambda = 1/6$ per second to each node, routing probability $p = 0.4$ and service times from the gamma(0.25, 4) model. The service-time variation is high, with coefficient of variation (CV) equal to 2. Overestimation of the mean queue length due to the renewal approximation is anticipated [6].

Fig. 4 plots the point estimates and 90% CIs for the steady-state mean queue length of node 5. The overestimation is apparent under Scenario A. The relative difference between the point estimate from Scenario A and that from the sequential approach is 10.71%. Modeling one more node on each side of

the neighbors of node 5 (Scenario B) drops the relative difference to 2.92%, a substantial improvement at a moderate extra cost for modeling two auxiliary nodes. This output accuracy can be considered adequate since the 90% CI from Scenario B overlaps with that from the sequential approach. Further augmentation of the buffered area increases accuracy in a diminishing manner: the relative differences for Scenarios C through E are 1.79%, 1.92%, and 0.78%, respectively (the increase from 1.79% to 1.92% is likely due to random error).

2) Case 1 (b)

This study focuses on the same heavily-loaded queueing network albeit with low service-time CV: $\lambda = 1/6$, $p = 0.4$, and gamma(4, 0.25) service-time distribution (CV = 0.5). Expected underestimation based on [6] is observed. The results are similar to Case 1 (a); see Fig. 5. The relative differences for Scenarios A through E are –9.65%, –1.54%, –1.25%, –0.73%, and –0.41%, respectively. We can conclude from Cases 1 (a) and (b) that the buffered-area mechanism is capable of reducing the estimation bias with little extra effort.

3) Case 1 (c)

In this study, the overestimation issue would be much less explicit than in Case 1 (a) since the load of the modeled network is light: $\lambda = 1/6$, $p = 0.2$, and gamma(0.25, 4) service-time distribution. Fig. 6 plots the estimates. The 90% CI from Scenario A overlaps with that from the sequential execution despite the small positive relative difference of 0.9%. Although the benefit of applying the buffered-area mechanism is modest, the mechanism does not cause any significant negative effect.

B. Case 2: 11×11 Grid-like Queueing Network

We explore the buffered-area mechanism performance with the two-dimensional (2D) grid-like expansion of the network in Case 1. The nodes in the 11×11 queueing network are labeled from 0 to 120 starting at the upper-left corner and going across from left to right (Fig. 2 (b)). Node 60 is the middle one. Each node contains a single-server, infinite-capacity queueing station with external Poisson arrivals of rate $\lambda = 1/6$ per second. The IID service times follow the gamma(0.25, 4) model. The served units move to the neighboring nodes with equal probability $p = 0.2$ or leave the network. The traffic intensities range from 0.35 to 0.82 (at nodes 0 and 60, respectively). We measure the steady-state mean queue-length estimates of node 60 with respect to various sizes and shapes of the buffered areas.

Given our focus on node 60, the network is partitioned into two portions: one consisting of the entire network but node 60 and the other with node 60 only. The buffered-area mechanism is adopted by the LPs modeling the latter part. The sizes and shapes of the buffered areas vary as illustrated in Fig. 7. No buffered area is applied in Scenario A. In Scenario B only two neighboring nodes of node 60 are added to the buffered area while in Scenario C all the four neighboring nodes are included; this makes the buffered area cross-shaped. Scenario D involves a square buffered area with the intention of showing that the square shape may be convenient for some simulation implementations despite the little gain in accuracy and the increase in simulation cost over Scenario C. In Scenario E, the square buffered area is expanded with one more node in every direction.

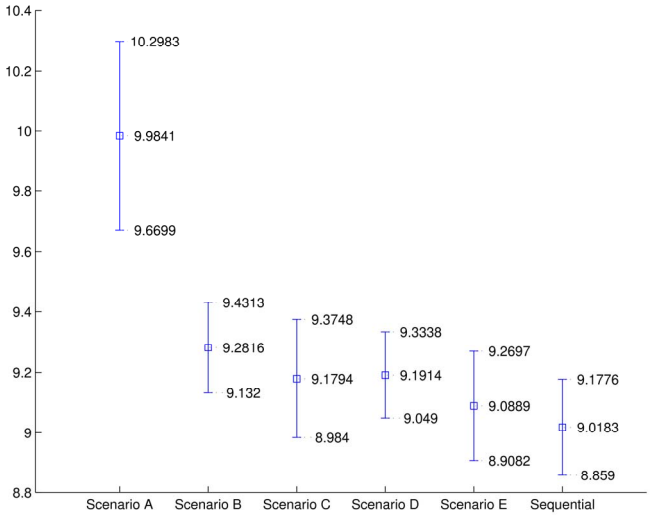


Figure 4. Point Estimates and 90% CIs for Steady-state Mean Queue Length of Node 5 in Case 1 (a)

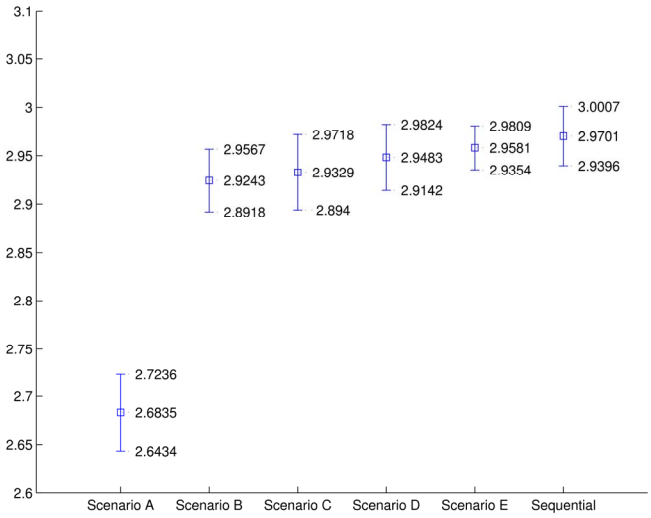


Figure 5. Point Estimates and 90% CIs for Steady-state Mean Queue Length of Node 5 in Case 1 (b)

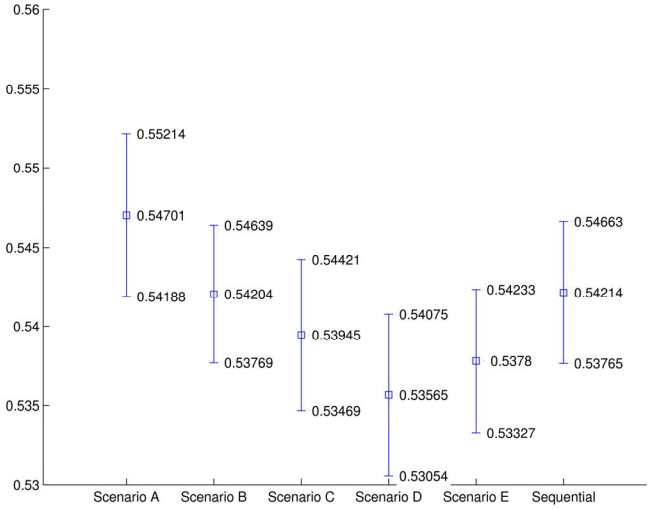


Figure 6. Point Estimates and 90% CIs for Steady-state Mean Queue Length of Node 5 in Case 1 (c)

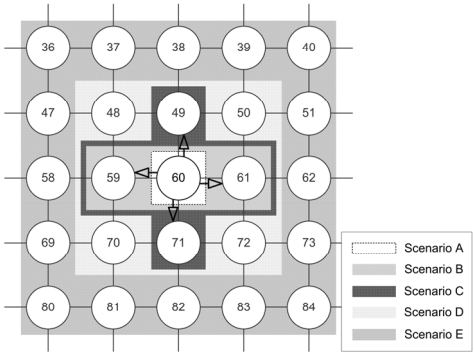


Figure 7. Application of Buffered-area Mechanism to 11×11 Grid-like Queueing Network

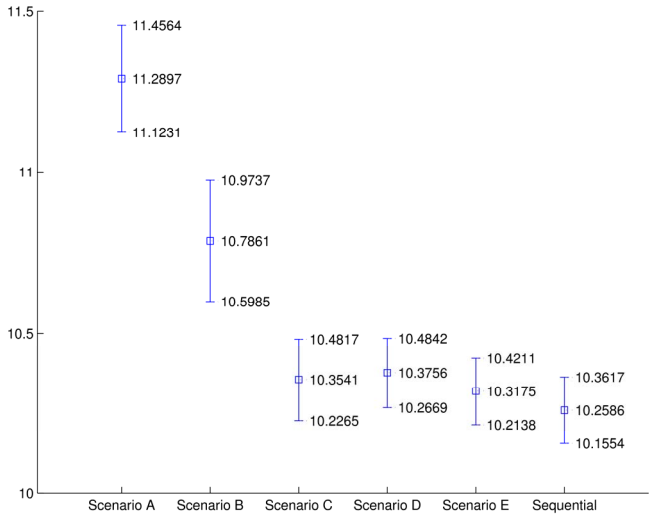


Figure 8. Point Estimates and 90% CIs for Steady-state Mean Queue Length of Node 60 in Case 2

The simulation settings of the ad hoc execution across the five scenarios are as follows. Ten independent experiments are performed. In each experiment every portion is modeled by 10 LPs. The results are compared against those from 100 independent sequential runs. Given the longer transient period for this complex queueing network (compared to the bidirectional tandem network in Case 1), the data collection pertinent to the estimates starts after 10 hours in simulation time and lasts for 30 hours.

Fig. 8 depicts the point estimates and 90% CIs for the steady-state mean queue length of node 60. The relative difference between the point estimate from Scenario A and that from the sequential approach is 10.05%. With neighboring nodes partially included in the buffered area (Scenario B), the relative difference drops to 5.14% while it drops further to 0.93% when all the four neighboring nodes are in the buffered area (Scenario C). As we anticipated, Scenario D reveals similar output accuracy as Scenario C; the relative difference for Scenario D is 1.14%. (The increase from 0.93% to 1.14% is likely due to random error.) Both the 90% CIs from Scenarios C and D overlap that from the sequential runs though some positive bias remains. Further expansion of the buffered area improves the output accuracy slightly as the relative difference for Scenario E is 0.57%. Based on Cases 1 and 2, it appears that the buffered area should contain at least all neighboring nodes of modeled subnetworks.

C. *Case 3: 8×8 Grid-like Queueing Network*

This experiment extends the study in [4] on an 8×8 grid-like queueing network with gamma(0.25, 4) service-time distributions. The network is partitioned into five 4×4 subnetworks: one located in the middle and each of the other four covering a corner; see Fig. 9. To apply the buffered-area mechanism we enlarge the modeled subnetworks to encompass the nodes immediately surrounding, as illustrated in Fig. 10. The areas in white are the original modeled subnetworks while those in grey represent the buffered area. All simulation configurations are the same as in [4] with the exception of the input links (the arrows with filled head in Fig. 10).

Fig. 11 charts the relative differences for the steady-state mean queue-length estimates of various nodes with or without the buffered-area mechanism. The output accuracy gain is apparent, especially for the nodes with high traffic intensity (e.g., nodes 18, 19, and 27). For example, the relative difference for the mean queue length at node 27 drops from 1.92% to 0.83%; at node 19 from 2.74% to 1.12%; and at node 18 from 1.93% to 0.12%. Despite the fact that the overestimation issue remains, the bias is reduced substantially: more than 100% improvement is observed in some nodes.

V. Conclusion and Future Work

Ad hoc distributed simulations offer a new approach for real-time system monitoring and control. In this paper we studied the applicability of the ad hoc approach to open queuing networks and proposed a buffered-area mechanism to improve the accuracy of system performance estimates. The experiments showed that the latter mechanism is able to reduce the bias of the steady-state mean queue-length estimates caused by the renewal approximations of arrival processes at nodes on the boundary of the modeled subnetworks. This reduction comes at a rather small increase in computational cost. For future work we plan to formulate the accuracy gain as a function of the size and shape of the buffered area. We also plan to study the overall execution efficiency of the ad hoc distributed simulation approach.

References

[1] R. Fujimoto, M. Hunter, J. Sirichoke, M. Palekar, H. Kim, and W. Suh, "Ad hoc distributed simulations," in Proceedings of the 21st International Workshop on Principles of Advanced and Distributed Simulation, IEEE Computer Society, pp. 15–24, June 2007.

[2] D. R. Jefferson, "Virtual time," ACM Transactions on Programming Languages and Systems, vol. 7(3), pp. 404–425, July 1985.

[3] Y.-L. Huang, C. Alexopoulos, M. Hunter, and R. M. Fujimoto, "Ad hoc distributed simulation of queueing networks," in Proceedings of the 24th International Workshop on Principles of Advanced and Distributed Simulation, IEEE Computer Society, pp. 57–64, May 2010.

[4] Y.-L. Huang, C. Alexopoulos, M. Hunter, and R. M. Fujimoto, "Ad hoc distributed simulation methodology for open queueing networks," Simulation, in press.

[5] W. Whitt, "The queueing network analyzer," The Bell System Technical Journal, vol. 62(9), pp. 2779–2815, November 1983.

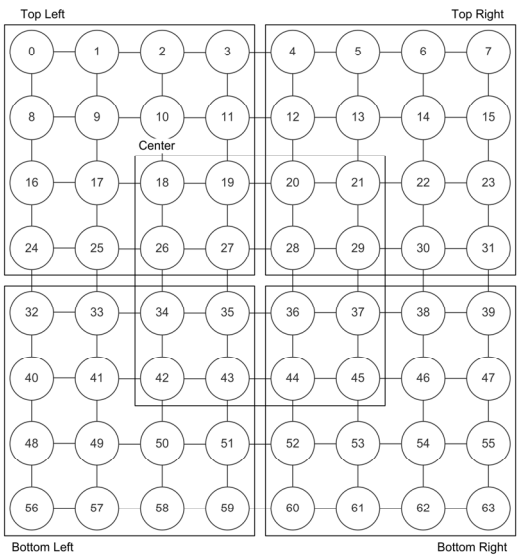
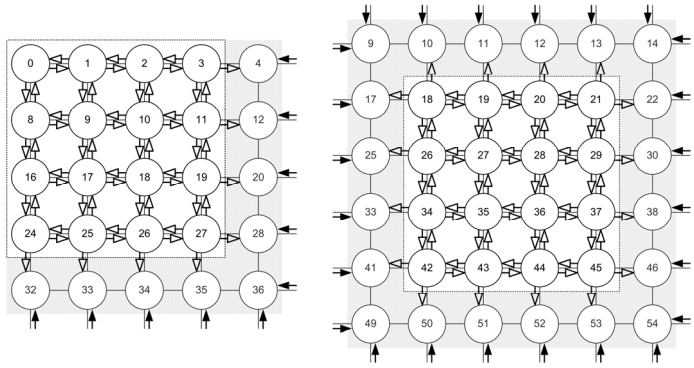


Figure 9. Partition of 8×8 Grid-like Queueing Network



(a) Top Left Portion with Buffered Area

(b) Center Portion with Buffered Area

Figure 10. Application of Buffered-area Mechanism to 8×8 Grid-like Queueing Network

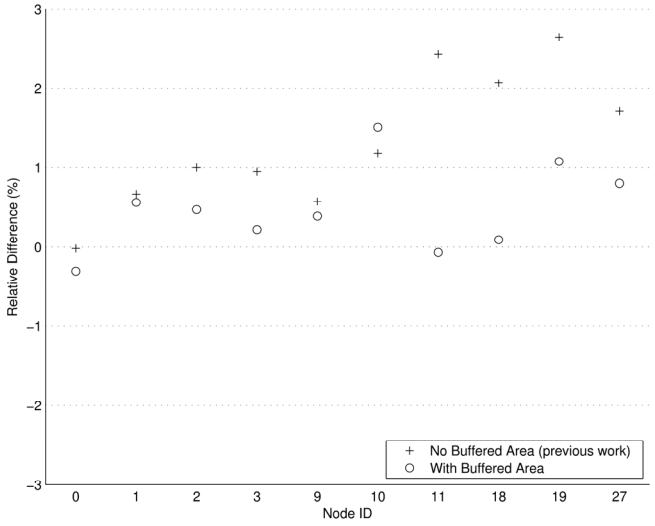


Figure 11. Relative Differences for Steady-state Mean Queue-length Estimates in Case 3

[6] L. N. Lester, "Accuracy of approximating queueing network departure processes with independent renewal processes," Information Processing Letters, vol. 16(1), pp. 43–48, January 1983.

[7] D. J. Daley, "The correlation structure of the output process of some single server queueing systems," The Annals of Mathematical Statistics, vol. 39(3), pp. 1007–1019, June 1968.

[8] D. J. Daley, "Queueing output processes," Advances in Applied Probability, vol. 8(2), pp. 395–415, June 1976.

[9] J. F. Reynolds, "The covariance structure of queues and related processes: a survey of recent work," Advances in Applied Probability, vol. 7(2), pp. 383–415, June 1975.

[10] K. B. Hendricks, "The output processes of serial production lines of exponential machines with finite buffers," Operations Research, vol. 40(6), pp. 1139–1147, December 1992.

[11] W. Whitt, "Approximating a point process by a renewal process, I: two basic methods," Operations Research, vol. 30(1), pp. 125–147, January–February 1982.

[12] S. L. Albin and S.-R. Kai, "Approximation for the departure process of a queue in a network," Naval Research Logistics Quarterly, vol. 33(1), pp. 129–143, February 1986.

[13] J. L. Jain and W. K. Grassmann, "Numerical solution for the departure process from the GI/G/1 queue," Computers & Operations Research, vol. 15(3), pp. 293–296, May 1988.

[14] S. G. Henderson, "Input model uncertainty: why do we care and what should we do about it," in Proceedings of the 2003 Winter Simulation Conference, IEEE Computer Society, pp. 90–100, December 2003.

[15] S. E. Chick, "Input distribution selection for simulation experiments: accounting for input uncertainty," Operations Research, vol. 49(5), pp. 744–758, October 2001.

[16] F. Zouaoui and J. R. Wilson, "Accounting for input model and parameter uncertainty in simulation," in Proceedings of the 2001 Winter Simulation Conference, IEEE Computer Society, pp. 290–299, December 2001.

[17] F. Zouaoui and J. R. Wilson, "Accounting for parameter uncertainty in simulation input modeling," in Proceedings of the 2001 Winter Simulation Conference, IEEE Computer Society, pp. 354–363, December 2001.

[18] R. R. Barton, B. L. Nelson, and W. Xie, "A framework for input uncertainty analysis," in Proceedings of the 2010 Winter Simulation Conference, IEEE Computer Society, pp. 1189–1198, December 2010.

[19] O. G. Batarseh and Y. Wang, "Reliable simulation with input uncertainties using an interval-based approach," in Proceedings of the 2008 Winter Simulation Conference, IEEE Computer Society, pp. 344–352, December 2008.

[20] P. Billingsley, Probability and Measure, 3rd Ed., John Wiley & Sons, New York, April 1995.

[21] C. Alexopoulos, "Statistical estimation in computer simulation," Chapter 8 in Elsevier Handbooks in Operations Research and Management Science: Simulation, S. G. Henderson and B. L. Nelson, Eds., Elsevier Science, Amsterdam, pp. 193–223, 2006.

[22] C. Alexopoulos, D. Goldsman, and R. F. Serfozo, "Stationary processes: statistical estimation," in Encyclopedia of Statistical Sciences, 2nd Ed., N. Balakrishnan, C. Read, and B. Vidakovic, Eds., John Wiley & Sons, Hoboken, NJ, pp. 7991–8006, 2006.

[23] R. L. Disney and D. Konig, "Queueing networks: a survey of their random processes," SIAM Review, vol. 27(3), pp. 335–403, September 1985.

[24] R. M. Fujimoto, "The virtual time machine," in Proceedings of the 1st Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, pp. 199–208, June 1989.

[25] K. Ghosh and R. M. Fujimoto, "Parallel discrete event simulation using space-time memory," in Proceedings of the 1991 International Conference on Parallel Processing, pp. 201–208, August 1991.

Application Transparent Migration of Simulation Objects with Generic Memory Layout

Sebastiano Peluso
DIS, Sapienza University of Rome
Italy

Diego Didona
INESC-ID, Lisbon
Portugal

Francesco Quaglia
DIS, Sapienza University of Rome
Italy

Abstract— **This paper presents the design of a global memory management architecture supporting application transparent migration of simulation objects (or LPs) whose state is scattered across dynamically allocated memory chunks. Our approach is based on a non-intrusive background protocol that provides each instance of the simulation kernel with information on the current mapping of the virtual address space of all the other instances. Dynamic memory requests by the application layer are then locally mapped onto virtual-address ranges that maximize the likelihood of being portable onto the address space of a remote kernel instance. In this way, independently of the load-balancing trigger (or policy), we maximize the likelihood that a desirable migration across a specific couple of kernels can actually take place due to compliance of the corresponding source/destination address spaces. We have integrated the global memory manager within the ROme OpTimistic Simulator (ROOT-Sim), namely a run-time environment based on the optimistic synchronization paradigm which automatically and transparently parallelizes the execution of event-handler based simulation programs conforming to ANSI-C. Further, we provide a contribution in the direction of widening load balancing schemes for optimistic simulation systems by defining migration triggers and selection policies for the objects to be migrated on the basis of memory usage patterns. An experimental assessment of the architecture and of memory oriented load balancing is also provided.**

I. Introduction

Well balanced load conditions have been shown to be mandatory for the effectiveness of parallel/distributed simulation systems, which is the reason why in literature a plethora of policies have been provided to dynamically redistribute the workload while the run is in progress (see, e.g., [1], [2], [3], [4], [5], [6]). However, beyond the definition of (or the reliance on) a specific load redistribution policy, load balancing techniques all entail the need for supporting the migration of simulation objects (or LPs) across different simulation kernel instances. This is a crucial aspect since depending on the facilities offered by the run-time platform, the migration protocol might require application level supports, e.g. for correctly setting up the state of the migrated object onto the destination kernel. On the other hand, having a run-time environment which is not able to provide full transparency for the migration of application level objects might discourage the use of load balancing facilities, which gives rise to a tradeoff that favors the application programmer's job at the expense of (unacceptably) reduced performance. For simulation environments based on technologies that allow the programmer to directly interact with the memory layout of the simulation objects (e.g. C/C++ technology), the above suboptimal tradeoff can easily become the de-facto reality, since the effort for developing ad-hoc application modules for relocating a generic-layout of the object state onto the destination address space might easily become prohibitive.

To address the aforementioned issue, we present the design of a global memory management architecture that allows completely transparent object migration across distinct simulation kernels (i.e. distinct address spaces). The architecture has been designed for integration within run-time environments based on C/POSIX technology, although the design indications remain valid even when considering other technologies, such as Windows systems. Our approach is based on a non-intrusive background protocol which provides each simulation kernel instance with a global view of the mapping of virtual addresses across all the kernels. In this way, dynamic memory allocation requests by the overlying layers are mapped onto ranges of virtual addresses that are re-mappable onto any distinct address space. This allows migrating objects without the need for explicit application level relocation, and with maximal flexibility across any couple of kernel instances. Such a flexibility can get reduced only in case of very huge virtual memory demand, since the likelihood of mapping a new range of virtual addresses that does not conflict with any other already mapped range within the whole system can get reduced. Fall back policies to deal with this kind of scenarios are discussed.

We also provide an integration of the global memory management architecture within the open source ROme OpTimistic Simulator (ROOT-Sim) [7], which is a C/POSIX based implementation of a run-time environment relying on the optimistic synchronization paradigm. Further, as an additional contribution of the paper, we introduce a memory triggered policy for load redistribution, and memory-pattern based metrics for selecting the objects to be migrated across different kernel instances. As we will show these can be used to complement classical (memory unaware) schemes for load balancing. An experimental assessment of the whole architecture and of the memory oriented load balancing policies is also provided.

The remainder of this paper is structured as follows. In Section II we provide the relation between our work and literature results. Section III provides the description of the global memory management architecture. In Section IV hints about the integration with the ROOT-Sim package are provided. The experimental study is presented in Section V.

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

II. Related Work

Since we deal with (transparent) migration in the context of parallel/distributed simulation systems, our work is related to all the literature solutions that have presented policies for load balancing in the context of either conservative (e.g. [1], [2]) or optimistic simulation (e.g. [3], [4], [5], [6]). The main difference between our proposal and these works resides in that they have been focused on the identification of load-balancing policies not tied to any specific architecture or platform. However, the validation of these policies has been typically supported via implementations not oriented to full transparency of the migration process, which is instead one core target of our work. According to this perspective, our architectural proposal can be considered as orthogonal and complementary to the above results.

On the other hand, when considering new hints we provide for the definition of memory-oriented triggers for load balancing in optimistic simulations, the closest work to our proposal is the one by Choe and Tropper in [8]. It presents a load characterization metric referred to as *processor space-time product*, where the space contribute consists in the amount of memory used to store events and states. While implementing load balancing schemes within ROOT-Sim, which are based on the transparent migration facilities provided by our architecture, we consider a metric that is meant to capture memory patterns and locality also in relation to the amount of productive work by each individual simulation object. This is rather different from the above metric, which is instead oriented to only capture memory demand globally caused by the whole set of hosted objects.

III. Global Memory Management

A. Overview

When dealing with application transparent migration of simulation objects with states exhibiting generic memory layout, the core issue to address is the management of *state-to-memory* binding. Specifically, in environments where the buffers belonging to the state layout can be dynamically allocated/deallocated by the application code, full transparency of migration facilities requires the coherence of references among virtual memory zones belonging to the state layout to be maintained upon migration between kernel instances. In other words, we have to preserve the positioning of the object state within the virtual address space of the destination kernel. This yields to a feasibility issue since installing a memory layout formed by predetermined virtual memory addresses within a different-process address space requires these addresses not to be currently mapped by that process. We tackle this problem via a global memory manager that validates and manages virtual addresses such in a way to guarantee the feasibility of migration with respect to the aforementioned constraint.

Generally, it is not necessary to respect this constraint for system level allocated buffers since simulation platforms entailing migration facilities typically embed mechanisms for relocating their own data structures (e.g. event-queues) into differently positioned buffers whenever these are involved in a migration operation. Consequently such a global memory manager needs to be designed to coexist with the pre-existing standard allocator operating within the local address space, namely the glibc/malloc library, which is still suited for kernel level buffers. Overall, our global memory manager is not aimed at serving memory requests by the simulation kernel. Hence, we logically see the whole address space as partitioned into two disjoint zones, namely application and kernel, whose relative sizes are not meant to be fixed, but can be dynamically modified. The application zone represents the virtual addressing range under the control of the global memory manager.

From an architectural perspective, the global memory manager is conceived as a software layer interposed between the operating system and the simulation kernel. Memory allocation/deallocation requests issued by the application layer can be trapped by the simulation kernel (e.g. via malloc hooking) and redirected towards the global memory manager. Internally, the global memory manager is formed by the following three components, layered as a stack:

- an *allocator*, representing the topmost element within the stack, which delivers currently unused memory areas by picking them from a pre-reserved (and hence already validated) pool;
- a *mapper*, which validates new memory areas when the *allocator* runs out of pre-reserved space;
- a *mapping policy*, which is responsible for choosing sets of contiguous and *migration suitable* addresses to be validated upon a request by the *mapper*.

The mapping policy is a distributed state machine based on a two-state representation, namely *conflict-free* and *conflicting*. Let $< k_1, \ldots, k_i, \ldots, k_n >$ be the n kernel instances involved in the simulation run, and let be A_i the set of virtual addresses already validated within k_i's address space (within the zone destined to the application), then the conflict-free state is characterized by the following predicate $\mathcal{P}_1$: $\forall h, j$ (with $h \in [1, n] \wedge j \in [1, n] \wedge h \neq j$), we have $A_h \cap A_j = \emptyset$. On the other hand, considering the above generic couple of distinct kernel indexes (h, j), the conflicting state is characterized by the predicate $\mathcal{P}_2$ (disjoint from $\mathcal{P}_1$): $A_h \cap A_j \neq \emptyset$.

By the above definition we have that, when residing within the conflict-free state, whichever memory buffer currently mapped by k_i can be re-mapped onto k_j by insisting on the same range of virtual addresses. In other words, migration of the content of that buffer from k_i to k_j does not require buffer relocation within k_j's address space.

The mapping policy is also in charge of providing each kernel k_i with consistent information related to A_j values associated with each other kernel. On the basis of this global view and the above two-state representation, memory validation requests coming from the *mapper* are served according to the following scheme, which is aimed at maintaining the global memory manager within the conflict-free state, so as to sustain maximal flexibility for buffers (and hence simulation object state images) migration over time:

- While being within the conflict-free state, a request from the *mapper* for n bytes on k_i is mapped to a range of n contiguous virtual memory addresses R_i such that the conflict-freedom property is still maintained when updating A_i to $A_i = A_i \cup R_i$. If this is not possible, the allocation is left pending and the conflicting state is entered.
- While being within the conflicting state, any (pending) request from the *mapper* for n bytes on k_i is mapped to a range of n contiguous virtual memory addresses R_i selected according to any locally feasible criterion, where infeasible means that we try to map the request onto already locally valid addresses. Among possible locally feasible criteria we suggest: i) to randomly pick the set R_i, or to leave the mapping decision to the operating system kernel, thus likely minimizing the cost for choosing a specific interval of addresses to be mapped; ii) to "steal" pages already validated by one or a subset of remote kernel instances, with the purpose of maintaining maximal flexibility for migration operations towards the remaining kernel instances.

We note that the lack of a global memory management scheme providing conflict-freedom (or controlled conflicts across subsets of kernel instances in case of huge virtual memory demand), would require application transparent migration to be necessarily supported via a protocol that, once selected a simulation object to be migrated from k_i to k_j, explicitly checks migration feasibility by verifying on-the-fly the compliance of the destination address space to the current object memory layout. As a consequence, once a re-balance decision is taken and the well suited set of objects to be migrated is identified (according to any policy), such a protocol would require an explicit handshake between source and destination kernels in order to verify the feasibility of the migration for the selected objects. This might induce additional latency to the migration protocol, which is instead avoided via global memory management, where each kernel has a global view of the current virtual memory mapping, and can select the objects to be migrated by a-priori avoiding migration of an object whose state layout is not compliant with the destination kernel memory map (which does never occur in case the global memory manager stationarily resides within the conflict-free state). In Section IV-B, we will provide a scheme exactly based on this approach. Also, in cases where the absence of a global memory manager yields to non-minimal likelihood of conflicts across the address spaces of different processes (e.g. due to biased selection of the virtual addresses validated by the local allocator), the above mentioned handshake could require to be executed iteratively in order to cope with planned migrations that are revealed infeasible while being actuated, unless the system gives up with the re-balance operation. In any case, additional loss in performance could be experienced. All the above drawbacks can be avoided at all via our global memory management protocol at the price of negligible overhead since, as it will be described in the next section, it can be implemented as a non-intrusive background protocol.

B. Implementation

1) The allocator: in our implementation, the *allocator* manages pre-allocated memory blocks according to the *segregated free list* scheme, in particular the *size classes with range lists* variation [9]. The elements in each list are stored in non-decreasing order of their size, with the purpose of implementing an $O(1)$ *first-fit* policy for the search of a free block. Moreover they are located within the same blocks so that the corresponding spatial overhead is fixed since it only consists of the directory of lists, made of a single pointer per list. All the additional management structures are embedded within each block in the form of a *header*, a *next* field and a *footer*. The header contains the size of the block and its status (allocated/free), as well as information about the status of contiguous memory blocks. The *next* field is a pointer that links a subsequent free block within the corresponding free list. Finally, the footer is used to allow *coalesce* operations [9].

2) The mapper: as hinted, the *mapper* validates sets of addresses predetermined by the *mapping policy* component and delivers them to the upper layer. For POSIX compliant systems, the support for this type of operation is provided by the `mmap` system call by raising the *MAP_FIXED* flag. Since `mmap` works according to page-based granularity, the page represents the unit for validation operations. Consequently, conflict-freedom for the global memory manager is instantiated in our implementation by considering A_i sets formed by pages rather than individual virtual addresses. This, in turn, implies that the minimal unit the *allocator* deals with becomes the page, and we demand the upper layers within the simulation kernel to exploit the allocation services by the global memory manager such in a way to avoid page sharing among distinct simulation objects, so that migration of a single page implies state-buffer migration for a single simulation object. One example of how to exploit page-based granularity provided by the *mapper*, still avoiding page sharing across different objects, will be provided while describing the integration of the global memory manager within the ROOT-Sim platform. In general, this can be achieved by simply reserving (on top of the allocator) a set of pages as the container of (variable size) chunks to be delivered to a specific simulation object.

The *MAP_FIXED* modality exploited by the *mapper* needs to be coherent with the memory layout management performed by the system, and aware that other libraries can use `mmap` too. To cope with this issue, we have aligned our implementation to memory management directives proper of the LINUX kernel 2.6.9 (which have been thoroughly revised compared to versions up to 2.4.x). According to these directives, the process stack has a fixed expansion boundary, starting from which `mmap` is exploited by the dynamic-library loader by validating addresses (i.e. pages) in a top-down fashion (namely from higher to lower virtual addresses). At the opposite side of the address space, the program break grows in a bottom-up fashion, starting from the process *data* segment. This

occurs under the control of the malloc library via the `brk` system call. However, for large allocation requests and/or when a page already mapped is encountered while moving forward the program break, the malloc library redirects the allocation request to the `mmap` service. Overall, the *heap* zone is in practice shared between the aforementioned system calls, which are both ultimately used by malloc, thus giving rise to the impossibility of clearly separating the relocatable data zone (i.e. the kernel level zone) from the one which is not (i.e. the application level zone). To address this issue, we have exploited the `mallopt` function provided by glibc [10] in order to force the malloc library to only rely on the `brk` system call. Our *mapper*, instead, allocates memory by exclusively relying on the `mmap` system call. However, to maintain memory allocation flexibility similar to the one originally provided by the malloc library, we have organized the mapper-zone (namely the memory zone destined to the application layer, which is under the control of the global memory manager) according to the following scheme (see Figure 1): (a) The start address of the mapper-zone is selected by moving x pages beyond the maximum *brk* value across all the kernel instances, as determined at loading time; (b) The end address of the mapper zone is set to be y pages above the lowest-address page allocated by the dynamic loader at loading time, across all the kernel instances.

The y non-validated pages after the end address of the mapper-zone are used as a buffer for the safe invocation of `mmap` by the dynamic linker, which typically works by not raising the *MAP_FIXED* flag, thus inducing a top-down validation scheme by the kernel (just insisting on those y pages). The x pages left as invalid after the *brk* value are used to allow memory allocation by the malloc library. Also, as we will show while presenting the *mapping policy* layer, the selection of migration suitable addresses to be mapped by the *mapper* tends to favor higher virtual memory addresses within the mapper-zone. This leaves the possibility to dynamically move the start address of the mapper-zone forward thus allowing the malloc library to further expand its allocation pattern within the heap, hence still supporting flexibility for memory allocation within the whole address space. Both x and y are parameters that can be configured at compile time.

3) The mapping-policy: the *mapping-policy* represents the core element within the global memory management architecture. It implements a distributed background-protocol that allows each kernel instance to detect the exact state of memory allocation of any other kernel, thus supporting global memory mapping rules proper of the conflict-free and conflicting states characterizing the state machine associated with the protocol.

In our implementation, such a distributed protocol is based on the token-ring approach. Therefore we have a token which circulates across all the local instances of the global memory manager (each on a separate simulation kernel k_i) and informs each of them about any change in the local page-mapping occurred on whichever other instance.

When the *mapping-policy* instance hosted by k_i receives the token, it reflects the updates piggy-backed onto the token

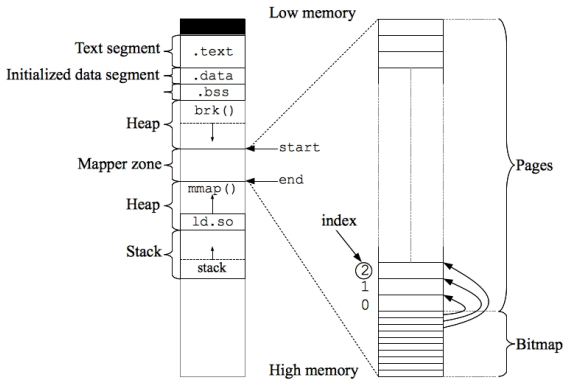


Fig. 1. Mapper-zone

within a local data structure. Hence, the instance hosted by k_i knows information about A_j values associated with mapped pages of any other instance. Similarly, the instance hosted by k_i can modify the local mapping A_i only when the token currently resides on k_i, so that any change in the local page-mapping is spread across all the other instances as if it happened within a global, distributed critical section.

The local data structure keeping track of the set of already validated pages by whichever kernel instance is a bitmap. This is stored within the highest-address pages within the mapper-zone, and every bitmap-block describes the state of one single page, specifying whether it is currently mapped and by which kernels (see Figure 1). To further optimize the management (transmission and processing) of the token, when the *mapping-policy* resides within the conflict-free state, the token payload only keeps track of the memory map updates occurred with the last cycle, and of the lowest address ever mapped within the mapper-zone by any *mapper* instance. We refer to this value as *global_brk*. By exploiting this value, permanence within the conflict-free state is guaranteed by simply mapping pages (whenever required) according to a top-down rule starting from the current value of *global_brk*. Specifically, upon the request by the *mapper* for validating x pages, the *mapping-policy* returns the set of addresses in the interval

$$[global\_brk - x \cdot PAGE\_SIZE, global\_brk)$$

and, contextually updates the token by piggy-backing the new value of *global_brk*.

When the request is such that the new value for *global_brk* would fall above the start address of the mapper-zone, the *mapping-policy* state-machine switches to the *conflicting* state. For handling memory allocations while in this state, we have decided to follow the previously illustrated stealing approach, where kernel k_i (currently allocating pages within its own address space) chooses a specific kernel k_j as the target for the stealing operation. This can be identified according to the following rule

$$j = \underset{h \in [1,n] \wedge h \neq i}{\operatorname{argmax}} \Psi(i,h) \text{ and } P(j,x) = true$$

where $P(j,x)$ is the predicate "k_j's *mapper* has validated at

least x contiguous pages or it has validated y contiguous pages next to z contiguous pages which are currently globally free such that $y+z \geq x$", and Ψ is a function which can be defined according to the adopted load balancing model in order to try to minimize the likelihood of infeasible migrations due to page conflicts across specific couples of kernels.

C. Optimizations

The reliance on the token-ring scheme to access the global critical section makes any invocation to the *mapper* potentially blocking, since the *mapping-policy* layer may need to wait for the arrival of the token in order to identify the range of virtual addresses to be mapped. This might be a problem especially when scaling up the size of the computing platform, since we may experience longer latency for the token arrival at each node. In order to address this problem, the global memory management architecture has been enriched with a *pre-mapping* scheme that is triggered when the amount of pre-reserved (and currently free) memory handled by the *allocator* layer falls under a predetermined threshold. In such a case, the *allocator* registers a callback function to support pre-mapping as soon as the next token arrival is experienced on the node. The pre-reserving scheme can be configured such in a way to select the amount of memory to be pre-reserved as a percentage of the amount of memory already mapped.

Finally, we have added another optimization which may help for small scale computing platforms and/or in scenarios where after a phase of memory expansion, the amount of virtual addresses required for the model execution remains relatively stable over time. For these scenarios, excessively frequent arrival of the token (likely occurring especially when the logical ring is small) might induce pure overhead, with no final benefit. Hence, we have implemented within the *mapping-policy* layer a mechanism that artificially delays the token circulation, by temporarily blocking the token at the logical master within the ring. A delay of 10 ms is applied by the master in case no change within the global allocation state is observed during the last token cycle. According to a TPC-like fashion, such a delay is dynamically increased (with step 10 ms) up to a `MAX_TOKEN_DELAY` value, whenever subsequent cycles still reveal no variation within the global allocation state. On the other hand, the delay is set to zero as soon as the token reveals to the master at least one change within the global allocation scheme.

IV. Integration within ROOT-Sim

In principle, our global memory management architecture can be integrated with any parallel/distributed simulation engine based on technologies compliant to those used for our project. This may occur independently of the specific nature (conservative vs optimistic) of the adopted synchronization protocol. In this section we focus on the integration of the global memory manager within ROOT-Sim so to enrich this run-time environment with transparent migration capabilities.

ROOT-Sim is an open source C/MPI-based simulation package targeted at POSIX systems [7], which implements a general-purpose parallel/distributed simulation environment relying on the optimistic (i.e. rollback based) synchronization paradigm. It offers a very simple programming model based on the classical notion of simulation-event handlers (both for processing events and for accessing a committed and globally consistent state image upon GVT calculations), to be implemented according to the ANSI-C standard, and transparently supports all the services required to parallelize the execution. It also offers a set of optimized protocols aimed at minimizing the run-time overhead by the platform, thus allowing for high performance and scalability.

As for management and recoverability of the state of simulation objects, which are crucial aspects for the design of effective optimistically synchronized environments, two main architectural approaches have been adopted. First, dynamic memory allocation and release by the application, performed via the standard malloc library, are *hooked* by the kernel and redirected to a wrapper. Second, the simulation platform is "*context-aware*", i.e., it has an internal state which distinguishes whether the current execution flow belongs to the application-level code or the platform's internals. In the former case, the hooked calls are redirected via the wrapper to an internal Memory Map Manager (called DyMeLoR), which handles per-simulation-object allocation/deallocation operations by maximizing memory locality of the state layout for each single simulation object, and by maintaining meta-data identifying the state memory map and making it correctly recoverable to past values (see [11]). In addition, the application level software can be compiled via an ad-hoc light instrumentation tool (see [12]) that transparently embeds a monitor module within the application, which provides the Memory Map Manager with the ability to track at runtime what memory areas (belonging to the current state layout) are modified. This facility is offered for compilation on IA-32/x86-64 architectures, and targets the standard ELF object file format. Through this feature, ROOT-Sim is able to transparently provide the application layer with both incremental and full checkpointing capabilities, each of which can better fit the specific application requirements. Such an instrumentation tool has been recently expanded (see [13]) in order to generate an executable version of the application layer based on a dual-coding approach, where two different versions of the application modules co-exist, one instrumented and one not. Thanks to the dual-coding approach, the ROOT-Sim kernel has been expanded in order to embed logics that are able to transparently switch between full and incremental log modes at run-time, also minimizing the overhead while executing application modules (since the non-instrumented version is run during any period where the full checkpointing mode has been selected as the one guaranteeing the best performance) [13].

The integration of the global memory manager within ROOT-Sim has been quite straightforward. Simply, the allocation requests by the object Memory Map Manager (DyMeLoR), which are executed on behalf of the application, have been redirected towards the global memory allocator, instead of the malloc library as in the original scheme (see

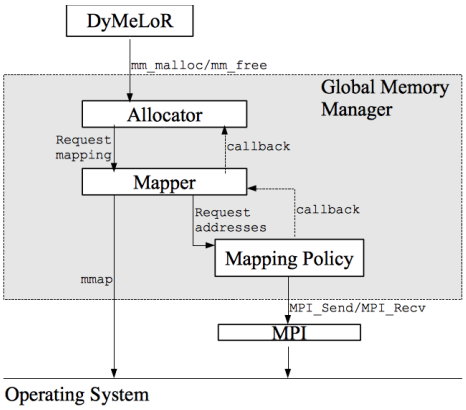


Fig. 2. Integration scheme

Figure 2). We note that this has been done with no intervention on the Memory Map Manager, which is therefore still in charge of (pre)allocating blocks of chunks for each specific object, and of keeping track of dirty chunks within the object memory map. The chunk sizes handled by the Memory Map Manager comply with those handled by the malloc library. Also, the Memory Map Manager has its own pre-reserving scheme such that, upon the first allocation request by a simulation object which fits a given chunk size, a block of memory able to contain 128 chunks is pre-allocated. In case it gets exhausted, a subsequent pre-allocation is executed, by doubling the amount of pre-allocated chunks. This approach straightforwardly fits the integration with the global memory manager, which guarantees per-simulation-object disjoint allocations with page-size minimal granularity. In fact, the Memory Map Manager will simply ask the global memory manager for a block of contiguous memory pages destined for chunk pre-allocation.

A. Migration manager

Migrating a simulation object across different ROOT-Sim instances involves transferring the associated event queues and also a sub-set of the logs currently belonging to the log-queue of the source kernel. In our design, we export a parameterizable number n of full-logs along the log chain ([1]), with the constraint that the most recent full log preceding GVT gets included in the migration. All the data transfer operations are based on packing multiple events/log-buffers onto MPI messages, so as to amortize the transmission set-up latency. We note that after an object is migrated, whichever migrated log can be safely reinstalled on the destination kernel (if needed due to the handling of rollbacks) since the memory chunks packed within the logs are guaranteed to belong (in terms of memory mapping) to memory blocks that have not been released for that simulation object by fossil collection operations. In fact, thanks to the underlying global memory manager, the state image of the migrating process is installed within the destination kernel address space by leaving unchanged the corresponding memory layout, for which no real memory release involved chunks that have not been released before the latest full checkpoint prior the GVT. However, once the migration operation is finalized, the corresponding memory pages on the source kernel get unmapped in order not to induce transition of the global memory manager into the conflicting state just due to migrations.

The *migration manager* is designed to provide non-blocking migration facilities, allowing the kernels not involved within the migration protocol to keep on processing events. While this mechanism is scalable, it makes necessary to enrich the messaging manager with a forwarding function to cope with messages destined to an object that is currently migrating. Moreover, the following properties need to be guaranteed:

i) Eventually all the kernel instances in the system are aware of a migration (hence will route events directly to the correct destination, which favors messaging latency).
ii) All in-transit messages are received by a migrated object before it can be involved in a new migration (otherwise, we incur the risk of not allowing the corresponding events/antievents to be eventually processed, which might even impact GVT advancement).

As for the property i), when the migration of a simulation object p is completed, the destination kernel k sends to all the others (except the migration source) a *MIGRATION_OK* message tagged with $< k, p >$. Upon the receipt of this message a *view change* occurs such that the local translation table that keeps the association between any simulation object and the corresponding hosting kernel gets updated. As for the property ii), a migrated simulation object is forbidden (*locked*) to migrate again until all in-transit messages (positive or negative) destined to it are reflected in its input queue. To achieve this goal, a kernel that performs a *view change* for a simulation object p sends an empty *event* message, tagged as *VIEW_CHANGED*, to p's sender. When the latter collects $n\_ker - 2$ of these messages for p, it sends a cumulative *VIEW_CHANGED* message to p's destination kernel. Finally, a kernel that receives a message tagged as *VIEW_CHANGED* for a local simulation object can unlock it, thus re-enabling future migrations. The correctness of the above protocol is guaranteed by the fact that MPI provides a *FIFO* point-to-point channel for messages with the same tag. Hence, when a kernel k receives a *VIEW_CHANGED* message m that is tagged as an *event* from a kernel k', all the previously sent messages along that point-to-point channel have been already flushed to the destination.

A final note is related to the treatment of in-transit messages associated with events generated by (or destined to) a migrating object. Given that MPI does not guarantee globally ordered communication over multiple channels, we might incur the situation where, e.g., a same destination object receives the events scheduled by the migrating object in reverse order, since they are injected from two different kernels. Beyond violating synchronization, which is anyway transparently recovered by

[1]Full logs for a given simulation object are taken sparsely even when the autonomic log/restore subsystem provided by ROOT-Sim switches to the incremental log mode for that object. This is done to optimize restore operations.

ROOT-Sim, this situation might give rise to inconsistences of the simulation model execution. To avoid this scenarios, we have enhanced the messaging system within ROOT-Sim by adding sequence numbers as tags on whichever message (positive or negative) across any couple of simulation objects, and we have implemented a delivery mechanism that guarantees ordering according to these sequence numbers.

B. The load balancing scheme

As for the policy determining load migration, we have implemented within ROOT-Sim a classical approach based on the observation of the percentage of productive simulation work by each kernel, and on the amount of virtual time advancement for Wall-Clock-Time unit (see, e.g., [4]). In particular, the base triggers for a re-balance decision are:

T1: $KPW_i > (1+\alpha) \cdot KPW_{mean}$, where KPW_i measures the ratio between CPU time for committed computation and CPU time for committed plus rolled back computation on kernel k_i.

T2: $WVTA_i > (1+\beta) \cdot WVTA_{mean}$, where $WVTA_i$ indicates the amount of Wall-Clock-Time required for the advancement of a simulation time unit on kernel k_i.

If one of the above two conditions is verified, then migration takes place by identifying those kernels k_j having the corresponding metrics over the threshold, and migrating simulation objects towards the kernels exhibiting the lower (under the mean) values for these metrics. Actually, in the current implementation, a single object is migrated across any couple of kernels (multi-migration across the same couple of kernels will be the object of future work). However, the above triggering conditions can be re-evaluated with frequency comparable (or even identical) to the one of the GVT protocol. Hence fast move of the workload across the kernels can be achieved in all the scenarios where the GVT protocol can be run frequently with minimal overhead, such as for tightly coupled platforms like multi/many core machines.

Beyond the above two triggers, we have also implemented an additional, secondary trigger oriented to optimize the behavior of the memory system. This additional trigger is defined on the basis of the below condition:

T3: $KMEM_i > (1+\gamma) KMEM_{mean}$, where $KMEM_i$ indicates the total amount of virtual memory used by the simulation kernel.

In this case, simulation objects are migrated towards the kernels that exhibit memory consumption under the mean. This might represent a relevant complementary trigger for optimistic synchronization based platforms since it is well known that these might incur performance degradation just due to the (very) large memory demand ([2]).

We have experimented two different approaches for re-balancing the load. For one approach load-balancing is triggered exclusively by T1 or T2 (i.e. T3 excluded), and the metric for the selection of the simulation objects to be migrated is the traditional efficiency factor introduced in [15] which is a good estimate of the object temporal efficiency. For object j, it is expressed as $\theta(j) = \frac{T_{commit}(j)}{T_{commit}(j)+T_{rollback}(j)}$ where $T_{commit}(j)$ and $T_{rollback}(j)$ express, respectively, the CPU time for committed and rolled back computation for this object. On the other hand, in the second configuration, load-balancing may also be triggered by the secondary, memory-oriented trigger T3, and we use a different metric to identify the objects to be migrated, which accounts for memory usage patterns. Specifically, we define an efficiency factor structured as $\theta'(j) = T_{commit}(j)/\sigma(j)$ where

$$\sigma(j) = \frac{peak\_log\_space(j) + committed\_events\_space(j)}{state\_size(j)}$$

The reverse of the efficiency parameter $1/\theta'(j)$ expresses the amount of virtual addressing span (normalized to the object state size) for one unit of CPU time used for productive work, which is meant to capture a kind of working set (in terms of virtual addresses) associated with the activities needed to support productive units of work by the object. Buffers for active (non-committed) events are not counted by $\theta'(j)$ since they will be considered within the space used for committed events by subsequent samples of $\sigma(j)$ while GVT advances.

On the basis of the above load-balancing triggers, the migration scheme operates at two levels:

Global: Periodically, the master kernel collects statistics about resources utilization from all the kernels, detects a possible unbalance condition by evaluating (T1 OR T2) and (secondarily) T3, and determines the kernels that will have to be involved in a migration.

Local: Upon the master's instructions, a kernel chosen to be the source in a migration selects the simulation object to be expelled.

In any case, the simulation object to be expelled is identified as one belonging to the subset of the local objects Γ, which is defined as $\Gamma(k_s) = P(k_s) \setminus \Omega(k_s)$, where $P(k_s)$ is the whole set of simulation objects hosted by k_s and $\Omega(k_s)$ is the one composed by the local simulation objects which verify at least one of the following conditions:

- they are currently *locked* by the migration protocol, thus waiting for the finalization of a previous migration;
- they have pages conflicting with the destination kernel (hence migration is actually infeasible);
- they are characterized by high communication degree with respect to local simulation objects. By "high degree" we mean that the ratio $\frac{messages_{local}}{messages_{remote}}$ is in the top $x\%$ within P (with x being a parameter defined at compile time). The contributions to this ratio are computed without considering the messages deleted by annihilation, in order to capture the real behavioral pattern of the interactions across the simulation objects.

Among the objects within Γ, we select the simulation object with the lowest efficiency value θ or θ', depending on the actual trigger for re-balancing the load, namely depending on

[2]Reverse computing approaches (see, e.g., [14]) have recently tackled this issue by reducing the memory demand for state-log buffers. However, the memory demand problem remains central, especially in contexts where reverse computing is not, or not easily, applicable.

whether T3 is active or not. Thus the source kernel gets rid of its most inefficient object, leaving more resources available for the virtuous ones.

C. Choice of rival kernels

The load balancing component is also responsible for exposing an interface to the *mapping-policy* which computes the function Ψ in order to provide locally the current *rival* kernels to steal pages from (whenever required). In the current implementation, the function Ψ is based on a clustering algorithm and, upon invocation, returns the closest kernel to the local one according to the metric *memory distance* which measures the difference in memory usage between two kernels. This function is conceived as a complementary block with respect to the secondary trigger T3, since the more two kernels are *memory distant* (thus likely being good candidates for being involved in a migration) the lesser there are chances to steal pages one from another. In order to avoid the scenario in which, according to this metric, a kernel k_i is rival of k_j but not viceversa, as soon as k_i steals a page from k_j, the latter is forced to choose the former as a rival too, which occurs via token based notification.

V. Experimental results

In this section we report experimental data for an assessment of both (A) the overhead due to the global memory management architecture, when compared with a traditional case where any allocation is driven by the malloc library, and (B) the effects on performance associated with the load-balancing schemes presented in Section IV-B, when operating on top of the global memory manager. As test-beds we have used a suite of differently parameterized simulation models of wireless communication systems adhering to GSM technology. The different parameterization entails variations of the transmission capabilities offered by each cell, as well as variations of the workload configuration (e.g. balanced vs non-balanced).

In the employed simulation models wireless communication channels are modeled in a high fidelity fashion via explicit simulation of power regulation/usage and interference/fading phenomena on the basis of the current state of the corresponding cell. Also, the power regulation model has been implemented according to the results in [16].

Upon the start of a call destined to a mobile device currently hosted by a given wireless cell, a call-setup record is instantiated via dynamically-allocated data structures, which gets linked to a list of already active records within that same cell. Each record gets released when the corresponding call ends or is handed-off towards an adjacent cell. In the latter case, a similar call-setup procedure is executed at the destination cell. Upon call-setup, power regulation is performed, which involves scanning the aforementioned list of records for computing the minimum transmission power allowing the current call-setup to achieve the threshold-level SIR value. Data structures keeping track of fading coefficients are also updated while scanning the list, according to a meteorological model defining climatic conditions (and related variations). The climatic model accounts for variations of the climatic conditions (e.g., the current wind speed) with a minimum time granularity of ten seconds. The employed simulation models have been developed for execution on top of ROOT-Sim in a way that each simulation object models a single wireless cell. Hence, the event-handler callback involves the update of individual cells' states, and cross-object events are essentially related to hand-offs between different cells.

The hardware architecture used for testing our proposal is a 64-bit NUMA machine equipped with two AMD Opteron 6174 processors and 32GB of RAM. Each processor has 12 cores that share a 12MB L3 cache, each core has a 512KB private L2 cache and 2200MHz speed. The operating system is 64-bit Suse Enterprise 11, with Linux Kernel version 2.6.32.13. The compiling and linking tools used are `gcc` 4.3.4 and binutils (`as` and `ld`) 2.20.0.

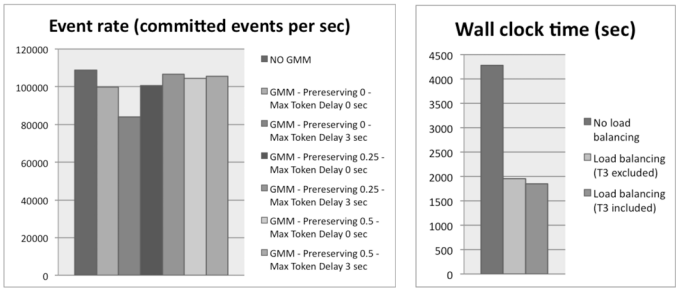
To evaluate the overhead due to the global memory management scheme we have performed a set of experiments where each cell sustains the same workload of incoming calls (hence we are in a load balanced situation), whose inter-arrival time is exponentially distributed, and whose average duration is set to 2 min. The expected rate for call inter-arrival has been set to achieve channel utilization factor on the order of 30%, while the residence time of an active device within a cell has a mean value of 5 min and follows the exponential distribution. For the above scenario, we have run experiments with 1024 wireless cells, modeled as hexagons covering a square region, each one managing 1000 wireless channels. We have performed runs where the load balancing triggers (T1, T2 and T3) are excluded, while the global memory manager is active. The latter has been also parameterized according to differentiated values for both pre-reserving (from 0.0 to 0.5 of the currently allocated amount of memory, with pre-reserving trigger set to 70% of the ratio between application delivered and total mapped memory) and the token maximum delay (from 0 to 3 sec). For all the configurations we have measured the event rate (expressed as the amount of committed events per Wall-Clock-Time unit), and we have compared it with the baseline case where allocation of memory exclusively occurs via the malloc library. As a final observation, the reported event rate samples have been computed as the average over 10 runs (parameterized with different seeds for pseudo-random generation). We have evaluated the event rate by explicitly taking samples related to the initial, transient phase of the simulation (namely 2500 complete calls per cell) where the overhead by the global memory manager is expected to be more evident due to the fact that the allocated memory is still being expanded towards a configuration where the same buffers are reused (redelivered) by the Memory Map Manager across sequences of deallocation/allocation operations by the application software. Also, with the employed setting the simulation model exhibits average event execution cost of about 70 microsec, which is relatively fine grain, hence representing a suited case study for evaluating the overhead of the global memory management architecture. By the results, shown in Figure 3(a), we see that when no pre-

reserving is adopted within the global memory management scheme ([3]), the baseline configuration exhibits performance gains up to 22%. However, as soon as the *mapper* performs at least a minimum amount of pre-reserving, the performance delivered by the global memory manager is comparable with that achieved by exclusively relying on the malloc library. This supports the feasibility of our proposal.

The above model has then been reconfigured in order to generate unbalanced workload across the wireless cells. In particular, for the 10% of the cells we have increased the call arrival rate in order to give rise to an increase of the average channel utilization factor by about 25% over the original value. Also, we have scaled-up CPU/memory requirements by increasing to 5000 the number of wireless channels handled by each cell (while maintaining the same utilization factor by also increasing the arrival frequency of calls). This has been done in order to provide an heavier configuration, which would allow us to better assess the effects of the employed load balancing policy. For this scenario we have evaluated the Wall-Clock-Time (averaged over 10 samples) for the execution of a minimal number N of calls on each cell, with N set to 20000 ([4]). For this settings we evaluated two different load-balancing policies (based on the triggers in Section IV-B), one excluding the memory-oriented trigger T3, and the other one including this trigger. The threshold parameters α, β and γ have been set to 0.1 and, concerning object selection for migration, we prevent migrating objects that exhibit internal vs external communication ratio which is ten times above the minimum value locally observed by the currently hosting kernel. Finally, GVT is recalculated each second, while the need for load re-balance is evaluated each five GVT cycles. The results are reported in Figure 3(b), where we also show the baseline case in which global memory management and load balancing are excluded. By the data we see that the proposed architecture allows reducing the Wall-Clock-Time by 50% or more, which is an indirect indication of limited overhead not only for the global memory manager (as discussed above), but also for the actual object migration protocol implemented within ROOT-Sim. Further, the inclusion of T3 as load re-balance trigger actually allows 4/5% further reduction of the Wall-Clock-Time. As a last note, these runs required on the order of 61E+06 events to be committed, with an average event cost of about 315 microsec. Hence, when considering a corresponding sequential execution based on an ideal, zero-latency scheduler, we would have observed speed-up of at least ten when the load-balancing scheme based on transparent migration is activated. Actual speedup values would be expected to be significantly larger, which indicates how the experimentation has been carried out when considering competitive parallel execution scenarios.

[3]In this case, pre-reserving is only performed by the Memory Map Manager and, as discussed in Section IV, operated at the level of blocks of memory got from the *allocator*, and destined to chunks to be delivered to a specific simulation object.

[4]Compared to the previous test cases, this time we have non-homogeneous event costs across the cells due to increase of the workload on a subset of them, which is the reason why we prefer Wall-Clock-Time as the performance indicator, compared to the event rate.



(a) Overhead evaluation for Global Memory Management - GMM - (1000 channels per cell) (b) Effects of load balancing (5000 channels per cell)

Fig. 3. Experimental data

REFERENCES

[1] A. Boukerche and S. K. Das, "Dynamic load balancing strategies for conservative parallel simulations," *Proc. of the 11th Workshop on Parallel and Distributed Simulation*, 1997, pp. 20–28.

[2] G. D'Angelo and M. Bracuto, "Distributed simulation of large-scale and detailed models." *International Journal of Simulation and Process Modelling*, vol. 5, no. 2, pp. 120–131, 2009.

[3] D. W. Glazer and C. Tropper, "On process migration and load balancing in time warp," *IEEE Trans. on Parallel and Distributed Systems*, vol. 4, no. 3, pp. 318–327, 1993.

[4] C. D. Carothers and R. Fujimoto, "Efficient Execution of Time Warp Programs on Heterogeneous, NOW Platforms," *IEEE Trans. on Parallel and Distributed Systems*, vol. 11, no. 3, pp. 299–317, 2000.

[5] P. L. Reiher and D. Jefferson, "Virtual time based dynamic load management in the time warp operating system," *Transactions of the Society for Computer Simulation*, vol. 7, pp. 103–111, 1990.

[6] C. T. Sina Meraji, Wei Zhang, "A multi-state q-learning approach for the dynamic load balancing of Time Warp," *Proc. of the 24th Workshop on Principles of Advanced and Distributed Simulation*, 2010, pp. 1–8.

[7] http://www.dis.uniroma1.it/~quaglia/software/ROOT-Sim/

[8] M. Choe and C. Tropper, "Flow control and dynamic load balancing in time warp," *Trans. Soc. Comput. Simul. Int.*, vol. 18, no. 1, pp. 9–23, 2001.

[9] P. R. Wilson, M. S. Johnstone, M. Neely, and D. Boles, "Dynamic storage allocation: A survey and critical review," *Proc. of the International Workshop on Memory Management*. Springer-Verlag, 1995, pp. 1–116.

[10] GNU, "The GNU C Library," http://www.gnu.org/software/libc/manual, March 2009.

[11] R. Toccaceli and F. Quaglia, "DyMeLoR: Dynamic memory logger and restorer library for optimistic simulation objects with generic memory layout," in *Proc. of the 22nd Workshop on Principles of Advanced and Distributed Simulation*. IEEE Computer Society, 2008, pp. 163–172.

[12] A. Pellegrini, R. Vitali, and F. Quaglia, "Di-DyMeLoR: Logging only dirty chunks for efficient management of dynamic memory based optimistic simulation objects," in *Proc. of the 23rd Workshop on Principles of Advanced and Distributed Simulation*. IEEE Computer Society, 2009, pp. 45–53.

[13] R. Vitali, A. Pellegrini, and F. Quaglia, "Autonomic log/restore for advanced optimistic simulation systems," *Proc. of the 18th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. IEEE Computer Society, 2010, pp. 319–327.

[14] C. D. Carothers, K. S. Perumalla, and R. Fujimoto, "Efficient optimistic parallel simulations using reverse computation," *ACM Trans. on Modeling and Computer Simulation*, vol. 9, no. 3, pp. 224–253, Jul. 1999.

[15] D. R. Jefferson, "Virtual time," *ACM Trans. on Programming Languages and Systems*, vol. 7, pp. 404–425, 1985.

[16] S. Kandukuri and S. Boyd, "Optimal power control in interference-limited fading wireless channels with outage-probability specifications," *IEEE Trans. on Wireless Communications*, vol. 1, no. 1, pp. 46–55, 2002.

Adaptive Message Clustering for Distributed Agent-Based Systems

Cole Sherer
Abhishek Gupta
Maria Hybinette

Computer Science Department
University of Georgia
Athens, GA 30602, USA

***Abstract*—Many agent-based simulation kernels rely on message passing in their core implementation. As the number of agents in a simulation increases or as the complexity of their communication expands the number of messages can increase exponentially. This is troublesome because the message content itself may be quite small, while the overhead, including message headers can dominate bandwidth and processing time. In these cases message passing becomes a bottleneck to scalability. The overhead of message exchange may saturate the network and degrade performance of the simulation. One approach to this challenge that has been investigated in related networking and simulation research centers is combining or "piggy-backing" multiple small messages together with a consolidated header. In many applications performance improves as larger, but fewer messages are sent. However, the pattern of message passing is different in the case of agent-based simulation (ABS), and this approach has not yet been explored for ABS systems.**

In this work we provide an overview of the design and implementation of a message piggy-backing approach for ABS systems using the SASSY platform. SASSY is a hybrid, large-scale distributed ABS system that provides an agent-based API on top of a PDES kernel. We provide a comparative performance evaluation for implementations in SASSY with a combined RMI and shared memory message passing approach. We also show performance of our new adaptive message clustering mechanism that clusters messages when advantageous and avoids clustering when the overhead of clustering dominates.

INDEX WORDS: Agent-Based Simulation, Distributed Simulation, Piggy-backing, Message Clustering, SASSY

I. Introduction

Agent-based simulation systems have gained significance in recent years because they provide a more natural way for developers to design their simulations than traditional discrete event simulators. ABS systems are applied in the study of multi-robot systems, social animal behavior, automobile traffic, and many other fields. The kernel of an ABS system implements communication, and a sense-think-act cycle using message passing. Because message passing is central to an ABS system, it may also become a bottleneck. The efficiency of this message passing is the focus of this research.

The topology of communication between agents depends on the simulation scenario. Some simulations may have agents with a limited sensing range, for instance ants have a limited sensing range in comparison to the size of an entire simulation environment (see Figure 1). In this case, the communication topology is simplified significantly from the fully connected case, which is often assumed in simulation implementations. Our approach is to exploit limitations to communication observed in real systems, such as limited sensing range or obstructions in terrain.

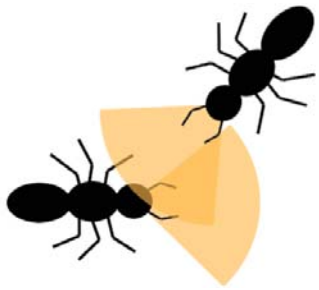


Fig. 1. Limited Sensing Range of Ants

As the complexity of ABS scenarios increase, through more agents or more communication, the number of messages exchanged may increase exponentially. This may degrade performance and result in a significant increase in execution time. Each message sent incurs overhead (e.g. processing headers, extracting messages from the network, and payload), and this research investigates methods to reduce the cost of exchanging messages and reduce redundant information. The aim of this research is to test the hypothesis that clustering messages before sending them can reduce the overall communication cost and improve performance. A clustering mechanism can improve performance by aggregating multiple messages into a single, larger message. We investigate factors that impact the effectiveness and performance of message clustering to speed up the performance of a distributed ABS and provide a quantitative analysis. We also propose a new adaptive scheme that dynamically determines when to utilize message aggregation and determine the number of messages clustered. We compare this with non-adaptive clustering and

978-1-4577-1365-1/11/$26.00 ©2011 IEEE

an unclustered approach.

Our empirical study and adaptive mechanism are implemented in the Scalable Agent-based Simulation SYstem (SASSY), a Java based distributed simulation kernel developed by the Distributed Simulation Lab at the University of Georgia [1]. SASSY is aimed to leverage advances in the field of parallel discrete event simulation (PDES) for agent-based simulations. It uses an optimistic synchronization protocol and was designed and developed to overcome the scalability issues that exist in current agent-based frameworks. Our implementation focuses on the Interest Management Logical Process (IMLP) mechanism implemented in SASSY [2]. Each agent in the simulation subscribes to a particular IMLP using a publish / subscribe protocol. The agents transmit messages and publish status updates via IMLPs. In our approach, messages to agents on remote machines are clustered at a local IMLP, forwarded to a remote IMLP, and then unclustered and sent to the receiving agent (see Figure 2).

The remainder of this paper is organized as follows: in the next section, we review related work from the artificial intelligence and simulation communities. In section III we present our implementation in detail. In section IV we present our results, and finally we conclude the paper with a discussion of future work.

II. Related Work

There has been some work done in the area to reduce communication costs in both the networking and simulation community. Our work is inspired by this previous work, but differs mainly in that it leverages knowledge available from the application layer and that it is adaptive in both the number of messages clustered and that it avoids clustering when it is determined advantageous. Our work is also unique in that it runs in a distributed agent-based simulation paradigm driven by a discrete-event simulation executive.

In the networking research community, Roy Friedman and Robbert van Renesse [3] compared the throughput and latency of two protocols (Tomfc and Dysfc) with and without message clustering. Their technique involved buffering the application messages for a short period of time and then sending them as a single packed message. Their study showed that message clustering improved both latency and throughput by two orders of magnitude. This improved performance was attributed to the fact that packing reduces the total number of bytes sent by replacing multiple packet headers with a single header for the clustered packet. This also causes less contention for network hardware, and fewer interrupts to handle messages.

Message aggregation techniques have also been proposed for sensor networks [4]. With the Application Independent Data Aggregation (AIDA) approach, He et al. made decisions based on the lower network layers instead of the application layer. Their approach has shown promising results for simulation. Other adaptive message aggregation protocols for sensor networks include RAP [5] and SPEED [6]. These protocols use the neighborhood (node) information like network congestion and traffic to make an informed decision about message routing and aggregation. SPIN [7] makes adaptive decisions to participate in data aggregation based on the cost of communication.

Other related techniques focus on aggregating data instead of messages. One of these techniques, Center at the Nearest Source (CNS) aggregates data at the source nearest to the destination [8]. In Shortest Path Trees (SPT), data is propagated along the shortest path from source to the destination and aggregated at common intermediate hops along the way.

Clustering of messages for agent-based systems in PDES is a relatively new concept. Takahashi and Mizuta addressed the message transmission costs in a large and complex agent-based simulation system on a large multi-node super computer, BlueGene, by clustering heavily communicating agents on the same node so that these agents can communicate via shared memory instead of remote message passing [9]. They essentially used a load-balancing approach to reduce communication costs. Their research shows promising performance results for 2 and 4 remote nodes connected via gigabit Ethernet. Our approach differs by using message clustering instead of load balancing to reduce communication costs. We leverage shared memory to communicate with agents that reside on the same node, and plan to incorporate load balancing in the future to further improve performance. A similar protocol in the networking community that promotes local communication is LEACH [10], a high-layer protocol that provides clustering and local processing to aggregate sensor data to reduce global communication.

Agents in our application use a publish and subscribe protocol between agents and Interest Managers (IMLPs). Each IMLP manages a subset of agents depending on where the agents reside in the environment [2]. Lees et al proposed a related approach where the world state was maintained by a tree of special logical processes known as Communication LPs (CLPs). The state of the environment is shared amongst these CLPs and any reads and writes to the environment state are facilitated by these processes. CLPs perform load balancing by swapping state variables when it is advantageous to reduce the total cost of access [11]. These researchers later studied rollback reduction by analyzing access patterns to CLPs. Since only certain accesses (e.g. premature reads) actually need to be rolled back, altering the Time Warp algorithm to take this into account reduces the total number of rollbacks (and thus the computation time) in a Multi Agent Simulation [12]. We intend to study these methods to reduce rollbacks in SASSY and further improve the performance of clustering.

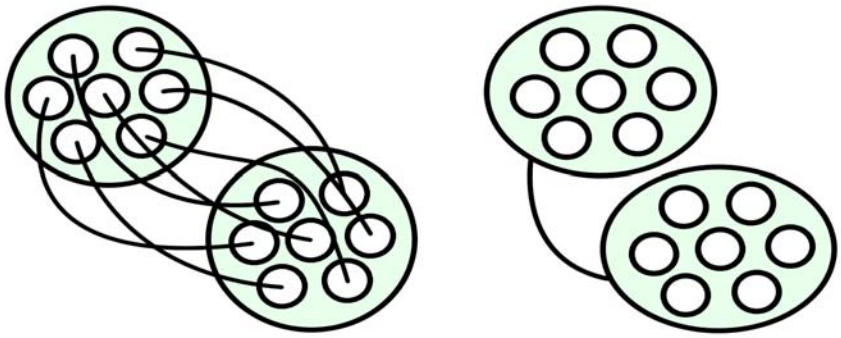


Fig. 2. Left: Agent Communication Across Grid Zones - Right: IMLP Message Clustering Approach

- Subscribe
- Unsubscribe
- Enter
- Leave
- Update
- **Ping**
- **Cluster**
- **Uncluster**

Fig. 3. SASSY Message Types - New Types are in Bold

III. Approach

To evaluate the effect of various message-clustering parameters on performance, we implement our own message-clustering scheme in SASSY and evaluate the scheme experimentally. In SASSY, agents are represented by a logical process (LP). These LPs maintain the most updated version of the environment visible to the agent. The SASSY middleware decomposes the sample space into a grid of cells to improve efficiency and scalability. An Interest Management Logical Process (IMLP) manages a cell or set of cells in the gridded environment. The IMLP implements a publish/subscribe system for LPs to ensure that all agents have a consistent view of their environment, unlike other agent-based systems that use a centralized global view of the environment for all agents [13], [14]. Every agent is mapped to at least one IMLP. We leverage this concept to implement our message-clustering algorithm, which is based on adding a few new message types into the existing SASSY system.

SASSY agents currently send five types of messages: subscribe, unsubscribe, enter, leave, and update. **Subscribe** messages allow LPs to monitor all messages that are published to a given IMLP. Similarly, the agent can send an **unsubscribe** message to stop receiving updates from a given IMLP. An **enter** message notifies the IMLP that an agent will be modifying the grid region that the IMLP manages. This may be something as simple as the agent moving through the region. Like an unsubscribe message, **leave** messages are sent by agents to inform an IMLP to stop listening for the given agent's updates. Once an agent enters an IMLPs grid space, it will send update messages that need to be relayed by the IMLP to all subscribing agents. To implement our message clustering mechanism, we added three new message types to SASSY (see Figure 3). **Ping** messages are sent amongst agents. The current IMLP that an agent is publishing information to will receive these **ping** messages and potentially cluster them. Clustered messages are created using a hash map to associate the destination LP with the message. This helps IMLPs route unclustered messages to the proper destination agents. Messages are clustered based on the destination agent's IMLP and then transmitted as one large **cluster** message. Once an IMLP receives a **cluster** message, it will uncluster the large message and forward the individual messages to the agents subscribed to it using an **uncluster** message. Agents will typically reside on the same machine as their IMLP, so this transfer can usually be done using shared memory instead of remote messaging. In previous work, Vulov, He and Hybinette showed that relaying messages through IMLPs showed better performance than sending messages directly between agents [15].

In this work we assume that agents generally multicast messages to a subset of agents in their environment instead of broadcasting to all agents. This is also true in biological systems such as ants which only communicate within their sensor range. The agents located in a cell at a particular instant will send messages only to agents in nearby cells. This ensures the message transmission is limited to multicast. To evaluate whether clustering messages in this environment is advantageous, we implemented two algorithms: fixed clustering and adaptive clustering. We compared these algorithms against a non-clustering approach. In the following subsections we will discuss these approaches.

A. Fixed Clustering

In the fixed clustering scheme a fixed number of messages (of variable payload size) are combined and then sent. There is also a configurable delay parameter that will cluster any buffered messages after a given period of time even if the cluster amount threshold has not been reached. This ensures

that all messages are delivered in a reasonable amount of time. This algorithm always clusters without taking parameters such as message size and network traffic into account. This behavior is depicted in Figure 4.

We observed that under some scenarios, clustering added a significant overhead. This overhead is addressed by our adaptive clustering mechanism that avoids clustering when it is not advantageous.

B. Adaptive Clustering

To avoid clustering overhead when advantageous, we propose an algorithm that adapts to the network traffic within the SASSY kernel. This is similar in spirit to the approach taken in AIDA [4] for sensor networks. Here the aim is to add an adaptive behavior that clusters when advantageous and avoids clustering otherwise.

To implement adaptive clustering we implemented message queues between the application layer and the SASSY kernel. All messages sent to the IMLP to be routed to respective agents must pass through this message queue. Our adaptive scheme monitors this message queue to ensure that it contains at least one message waiting to be processed by the kernel. If so, it infers that the network traffic is high and performs clustering. However, in scenarios where we observe an empty message queue, the kernel will transmit messages in their original form without aggregation. This approach gave us an adaptive clustering mechanism based on internal network traffic. When this approach was compared with fixed clustering we found that for equal threshold and end-to-end delay values the adaptive approach performed much better than the fixed one.

IV. Results

We performed four experiments to show the benefits of clustering in an agent-based simulation system. We first determined that sending fewer, larger messages is more efficient than lots of small messages. Based on these results, we developed a clustering approach and tested it against the unclustered approach by altering the number of agents in the simulation and the message payload sizes. In our final experiment, we tested the performance of the adaptive clustering mechanism over fixed clustering. All experiments were conducted in a distributed environment with 10 IMLPs, 10 PEs, 50 - 500 agents and message payload sizes ranging from 1 - 8000 bytes. Tests were executed on dual-core 2.6 GHz workstations with 4GB of RAM and networked with gigabit Ethernet. The results of our experiments are detailed in the following subsections.

A. Message Clustering

Our first experiment was conducted to assess the impact on communication overhead resulting from sending large packets as opposed to several smaller packets. This experiment was conducted using 10 IMLPs spread across 10 PEs with 500 agents. To understand the message clustering impact we selected values that depend on payload size. The size of message payloads was fixed, and the experiment was repeated four times sending a variable number of fixed-size messages each run. In these runs, the total payload was fixed and split into multiple smaller messages. The results are shown in Figure 5. The experiment was repeated with four different message sizes from 1KB to 8KB. As expected, for each message size it was much faster to send a single packed message instead of multiple smaller messages. This demonstrated that clustering could potentially improve run times in agent-based simulations.

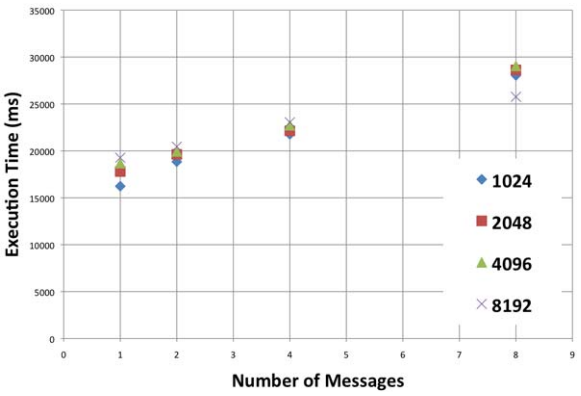


Fig. 5. For each size, a single message ran faster than multiple smaller messages (lower is better)

B. Varying Numbers of Agents

In our next experiment, we began to explore a clustered approach. Experiments were run on 10 machines (one PE per machine). We varied the number of agents in the system and each agent sent 50 messages that were randomly sized between 1 and 1000 bytes. After sending each message, agents would wait a random amount of time up to 10ms to simulate computation. Quantitative results for this experiment are shown in Figure 6. The X-axis shows the number of agents and the Y-axis shows the execution time in milliseconds. Each data point represents the average execution time over five runs. The clustered approach was initially faster than the unclustered approach, but as the number of agents in the system increase, the two approaches converge. We ran an additional trial with 1000 agents and found that the unclustered approach performed better. We believed this to be a result of the increased rollbacks caused by too many agents assigned to each Interest Manager. We repeated this trial with 100 IMLPs instead of 10. With a lower agent-to-IMLP ratio, the clustered approach performed with a speedup of 1.8 over the unclustered one.

C. Varying Message Size

The next experiment was to highlight scenarios when the message clustering approach may not be advantageous and to

```
private void PING(Message msg)
{
    numClustered++;
    destination = imlpFor(msg.getAPPid());

    if(buffer.containsKey(destination))  {
        buffer.get(destination).add(msg.getMyMessage());
    }
    else  {
        buffer.put(destination, addMessage(msg.getMyMessage()));
    }

    if(numClustered > threshold || currentLocalTime > maxClusterTime)  {
        for(String destination : buffer.getKeys())  {
            sendMessage(buffer.get(destination), destination);
        }
        resetBufferAndClusterTime();
    }
}
```

Fig. 4. Fixed Clustering Approach

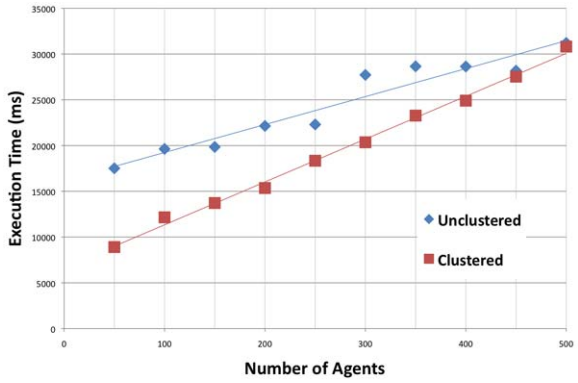


Fig. 6. Runtime vs. Number of Agents on 10 IMLPs (lower is better)

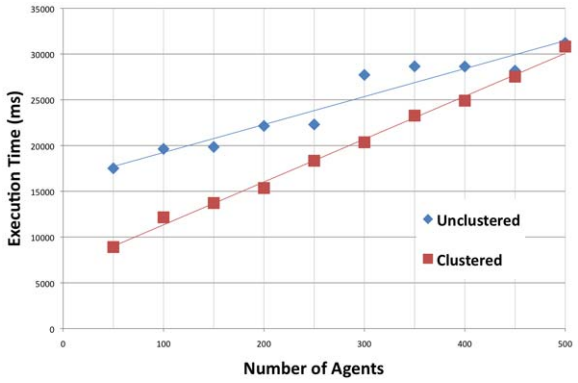


Fig. 7. Runtime vs. Message Size (lower is better)

give us insight into how to parameterize adaptive clustering. To study this we ran multiple iterations of fixed clustering with message sizes between 20 and 500 bytes. We measured many small incremental changes in message size to observe the impact of message clustering with smaller messages and note the threshold value at which clustering starts to impact the system performance. This experiment was conducted on 10 PEs, 10 IMLPs and 500 agents. The results of this experiment are shown in Figure 7. We observed that message size does not have a significant impact on execution time in either method. As you can see, each method takes approximately the same amount of time independent of message size. This shows that the number of messages sent, and the number of agents in the system have a greater impact on execution time than message size.

D. Adaptive vs. Fixed Clustering

Our final experiment was conducted to test our adaptive clustering approach. There are a number of factors that impact the performance of message transmission including message size and processing time, latency, bandwidth and external work loads. To test the adaptive clustering approach we focused on number of agents. We varied payload from 1 - 1000 bytes, and number of agents from 50 - 500. The experiment was distributed over 10 PEs with 10 IMLPs. Once again the results in Figure 8 represent the average of 5 runs. Adaptive clustering showed much more speedup than fixed clustering. The adaptive clustering approach takes into account the internal traffic that depends on the processing time of the SASSY kernel and also the time taken to copy messages between network and local storage. Clustering is performed when the queue that

holds unprocessed messages contains unsent messages. Hence we observe that for the same amount of delay, performing clustering of messages was not always helpful compared to the adaptive approach.

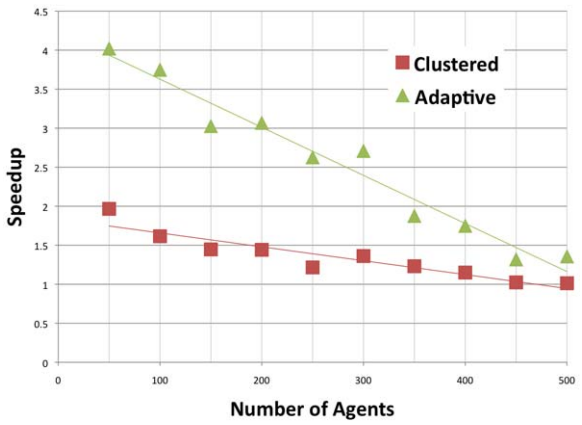


Fig. 8. Speedup of Adaptive and Fixed Clustering vs Unclustered Approach (higher is better)

V. Conclusions and Future Work

In this work we have shown that message clustering, or "piggybacking," can improve performance in large scale distributed agent-based simulation. We demonstrated the benefit of message clustering in a PDES system. We evaluated traditional messaging as well as fixed and adaptive clustering approaches by implementing them in SASSY, a Java agent-based PDES system. Our results indicate that the decision to cluster should depend on message payload size and number of agents in the system. Fixed clustering may not always be beneficial, like when the agent LP to IMLP ratio is high, and the overhead of clustering and unclustering messages dominates. The adaptive clustering approach monitors the system to determine when clustering will be beneficial. This approach always outperforms both the unclustered approach and fixed clustering approaches.

This research restricts agents to a single processing element (PE). To achieve a more realistic agent-based scenario, we would like the agents to be able to migrate across PE nodes dynamically. This would allow us to investigate communication patterns for complex, realistic agent-based systems such as colonies of ants or bees. Movement across nodes would also help us distribute the load of the simulation system by aggregating highly interactive agents onto a single node so that they could communicate via shared memory instead of sockets. This would help us investigate the effect of external factors like load balancing on our adaptive clustering technique.

Our clustering approach was always tested with a threshold of one time step. If an Interest Manager received any messages that were at least one time step ahead of local time, the clustered messages would immediately be sent and new clusters started. In the future we will explore the effect of longer threshold times on rollbacks and execution time.

References

[1] M. Hybinette, E. Kraemer, Y. Xiong, G. Matthews, and J. Ahmed, "SASSY: A design for a scalable agent-based simulation system using a distributed discrete event infrastructure," in *Proceedings of the 2006 Winter Simulation Conference*, Dec. 2006, pp. 926–933.

[2] T. He and M. Hybinette, "A comparison of interest manager mechanisms for agent-based simulations using a time warp executive," in *pads-08*. IEEE Computer Society, 2008, pp. 157–162. [Online]. Available: http://doi.acm.org/10.1145/1381309.1382304

[3] R. Friedman and R. van Renesse, "Packing messages as a tool for boosting the performance of total ordering protocols," in *Proceedings 6th International Symposium on High Performance Distributed Computing (HPDC '97) (6th HPDC'97)*. Portland, OR, USA: IEEE Computer Society, Aug. 1997, pp. 233–242.

[4] T. He, B. M. Blum, J. A. Stankovic, and T. Abdelzaher, "AIDA: Adaptive application-independent data aggregation in wireless sensor networks," *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 2, pp. 426–457, May 2004.

[5] C. Lu, B. M. Blum, T. F. Abdelzaher, J. A. Stankovic, and T. He, "RAP: A real-time communication architecture for large-scale wireless sensor networks," in *IEEE Real-Time and Embedded Technology and Applications Symposium*. IEEE Computer Society, 2002, pp. 55–66. [Online]. Available: http://computer.org/proceedings/rtas/1739/17390055abs.htm

[6] T. He, J. A. Stankovic, C. Lu, and T. F. Abdelzaher, "SPEED: A stateless protocol for real-time communication in sensor networks," in *International Conference on Distributed Computing Systems (ICDCS-2003)*. IEEE Computer Society, 2003, p. 46.

[7] W. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive protocols for information dissemination in wireless sensor networks," in *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. ACM, 1999, pp. 174–185.

[8] C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann, "Impact of network density on data aggregation in wireless sensor networks," in *Proceedings of the 22nd International Conference on Distributed Computing Systems*. IEEE, 2002, pp. 457–458.

[9] T. Takahashi and H. Mizuta, "Efficient agent-based simulation framework for multi-node supercomputers," in *Winter Simulation Conference*, L. F. Perrone, B. Lawson, J. Liu, and F. P. Wieland, Eds. WSC, 2006, pp. 919–925. [Online]. Available: http://doi.acm.org/10.1145/1218112.1218281

[10] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS '00)*, 2000. [Online]. Available: http://computer.org/proceedings/hicss/0493/04938/04938020abs.htm

[11] M. Lees, B. Logan, R. Minson, T. Oguara, and G. K. Theodoropoulos, "Distributed simulation of MAS," in *Multi-Agent and Multi-Agent-Based Simulation, Joint Workshop (MABS 2004)*, ser. Lecture Notes in Computer Science, P. Davidsson, B. Logan, and K. Takadama, Eds., vol. 3415. Springer, 2004, pp. 25–36.

[12] M. Lees, B. Logan, and G. K. Theodoropoulos, "Using access patterns to analyze the performance of optimistic synchronization algorithms in simulations of MAS," *Simulation*, vol. 84, no. 10-11, pp. 481–492, 2008. [Online]. Available: http://dx.doi.org/10.1177/0037549708096691

[13] P. F. Riley and G. F. Riley, "SPADES – a distributed agent simulation environment with software-in-the-loop execution," in *Proceedings of the 2003 Winter Simulation Conference (WSC-2003)*, Dec. 2003, pp. 817–825.

[14] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the International Conference on Advanced Robotics*, Coimbra, Portugal, Jul 2003, pp. 317–323.

[15] G. Vulov, T. He, and M. Hybinette, "Quantitative assessment of an agent-based simulation on a time warp executive," in *Proceedings of the 2008 Winter Simulation Conference(WSC-2008)*, S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, and J. W. Fowler, Eds. WSC, 2008, pp. 1068–1076. [Online]. Available: http://dx.doi.org/10.1109/WSC.2008.4736175

Author Index