Towards Performance Evaluation of Conservative Distributed Discrete-Event Network Simulations Using Second-Order Simulation

Philipp Andelfinger and Hannes Hartenstein
Steinbuch Centre for Computing and Institute of Telematics
Karlsruhe Institute of Technology
76131 Karlsruhe, Germany
{philipp.andelfinger, hannes.hartenstein}@kit.edu

ABSTRACT

Whether a given simulation model of a computer network will benefit from parallelization is difficult to determine in advance, complicated by the fact that hardware properties of the simulation execution environment can substantially affect the execution time of a given simulation. We describe SONSim, an approach to predict the execution time based on a *simulation of an envisioned distributed network simulation* (second-order simulation). SONSim takes into account both network model characteristics and hardware properties of the simulation execution environment. To show that a SONSim prototype is able to predict distributed performance with acceptable accuracy, we study three reference network simulation models differing fundamentally in topology and levels of model detail – simple topologies comprised of interconnected subnetworks, peer-to-peer networks and wireless networks. We evaluate the performance predictions for multiple configurations by comparing predictions for the three reference network models to execution time measurements of distributed simulations on physical hardware using both Ethernet and InfiniBand interconnects. In addition, utilizing the freedom to vary simulation hardware and model parameters in the second-order simulation, we demonstrate how SONSim can be used to identify general model characteristics that determine distributed simulation performance.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems—*Modeling Techniques*; I.6.1 [**Computing Methodology**]: Simulation Theory—*Model Classification*; I.6.5 [**Computing Methodology**]: Model Development—*Modeling Methodology*; I.6.8 [**Computing Methodology**]: Types of Simulation—*Parallel, Distributed, Discrete Event*

SIGSIM-PADS'13, May 19–22, 2013, Montréal, Québec, Canada.

Keywords

performance prediction, simulation, parallel, distributed, discrete-event, network simulation, network models

1. INTRODUCTION

Simulation is used to investigate computer networks that are both difficult to model analytically and that cannot be examined physically. However, accurate simulation of complex and large networks is time-consuming, creating the need for performance improvements. Parallel and distributed simulation can potentially reduce runtime by distributing the computational load to multiple processors connected by a network or by shared memory. However, the benefit of parallelization of network simulations described in the literature varies: in some cases, a substantial reduction of runtime and near-linear scaling with the number of processors was achieved [12, 24]. In other cases, only limited speedup was observed compared to sequential simulation, or performance gains did not scale beyond modest numbers of processors [16, 22]. Development of an efficient parallel implementation of a simulation is both time-intensive and error-prone [3]. Therefore, to avoid unnecessary effort, the benefit obtained by parallelization of a simulation should be evaluated prior to implementation. Additionally, there is a need to reconsider the performance of distributed simulations with respect to modern elastic execution environments exemplified by infrastructure-as-a-service offerings [31]. Particularly when considering usage-based billing, the issue of choosing an efficient scale for distributed simulations becomes a critical one. Furthermore, a general classification of network models according to their parallelization potential would be helpful.

In this paper, we describe *SONSim* (**s**econd-**o**rder **n**etwork **sim**ulation), an approach to predict the benefit gained through parallelization of simulations of computer networks. Based on information gathered during the runtime of an existing sequential simulator, SONSim simulates distributed execution on sets of interconnected processors, factoring in both properties of the simulated network and the hardware to be used for simulation. We refer to simulation of a simulation as *second-order simulation*.

The SONSim approach is a first step to attack the problem of deciding whether the effort for parallelization of a discrete-event network simulation is justified, based solely on properties of an existing sequential simulator and measurements of the network the simulation is to be executed in. Additionally, if a sufficient benefit cannot be expected

for a given model, SONSim predicts the impact of modifications to the network model and to the hardware used on distributed simulation performance. More generally, by allowing modelers to vary properties of a network model and the hardware, SONSim will help determining not only *whether* a given simulation will benefit from parallelization, but can also give hints towards *why* this is the case.

Utilizing SONSim's performance predictions and the ability to vary key model parameters, we present examples on how SONSim can enable more general statements on the potential of classes of network models for parallelization.

The rest of the paper is structured as follows. In Section 2, we discuss related work. Section 3 describes the methodology underlying SONSim's operation and specifies the steps required to obtain performance predictions as a basis for investigating performance characteristics of computer network models. Section 4 characterizes the experiments conducted. Section 5 discusses the results and the prediction accuracy. In Section 6, we investigate the distributed performance of simulations under varying model detail, lookahead and simulation scale, demonstrating the usage of SONSim for making more general statements about the distributed performance of network simulations. Finally, Section 7 gives a summary of our work.

2. RELATED WORK

The performance of parallel and distributed simulations has been the focus of intensive research for over twenty years. In the past few years, there has been a renewed interest in performance investigation of parallel applications in general [14, 27] and parallel and distributed simulations in particular [10, 31], as current multi-core and many-core architectures make parallel processing not a preference, but a requirement for steady increases in computational performance [29]. Our focus is on *discrete-event* simulations characterized by system state which is manipulated by events scheduled at discrete points in simulated time, focusing on conservative synchronization where simulation correctness is ensured at all times by a synchronization algorithm.

Existing works in the field of performance prediction for parallel and distributed simulations can be grouped into three categories: *measurements*, *analytical modeling* and *simulation*. As we focus on the performance resulting from interactions between properties of the simulated networks and the hardware used to run the simulation, we disregard the wide range of work studying the performance of specific synchronization algorithms and simulator implementations for this overview.

First, *measurements* have been widely used to determine the potential and real-world benefits of parallel discrete-event simulations [5, 18, 23] with respect to specific hardware, simulator implementations and synchronization algorithms. Bagrodia et al. [2] described a performance evaluation method for comparing synchronization algorithms. An execution trace is generated during runtime of a sequential simulator and subsequently used to guide a parallel simulation. The execution trace allows for global knowledge of the simulation progress and enables perfect synchronization. In a recent work, De Munck et al. [10] studied the performance of parallel and distributed simulations in modern multi-core environments using traditional synchronization algorithms and proposed hybrid and adaptive algorithms to reduce synchronization overheads. The algorithms were evaluated with reference to simulation runs based on execution traces. In our work, guided parallel runs on physical hardware are also used for the evaluation of SONSim's prediction accuracy. However, for the performance prediction itself, instead of executing a simulation using physical hardware, SONSim simulates the execution. Therefore, a parallel implementation of the simulator is not required and properties of the simulation model and the hardware used for the simulation can be varied easily.

Second, ideally, the system under study can be accurately modeled *analytically*, allowing for mathematical predictions of distributed performance. Critical path analysis [7] is a common analytical approach which determines the parallelism inherent in a simulation based on an event precedence graph representing event interdependencies. Critical path analysis determines an upper bound for the speedup achievable for a given simulation model using conservative synchronization, but in contrast to SONSim does not explicitly consider the hardware in use.

Some works combine measurements and analytical considerations to perform performance predictions. Gianni et al. [13] utilize knowledge of the hardware to be used and design-time information about the model to conduct predictions based on queuing networks. In 1999, Liu et al. [19] used microbenchmarks performed on physical hardware as a basis for back-of-the-envelope calculations to give surprisingly accurate predictions of parallel performance. In general, due to the complex interactions between real-world models and the used hardware, mathematical performance analysis is only feasible with reasonable effort when applying simplifications, resulting in a trade-off to be made between performance prediction accuracy and analytical effort. Like critical path analysis, SONSim employs simplifications to help understand a model's inherent parallelism without being closely tied to hardware specifics. However, as the hardware used for the distributed simulation does have substantial impact on performance, we consider it explicitly.

Third, *simulative* approaches base their predictions on measurements performed during simulator runs on physical hardware in combination with subsequent simulations of the parallel simulation under study. These approaches are similar in nature to performance prediction methods for general applications [6]. However, the discrete-event paradigm limits the entities and activities involved in modeling parallel simulations, allowing for less complex models. As in the existing works using simulative approaches for performance prediction of parallel simulations, we use traces of sequential simulation runs to guide the predictions. Using this approach, both hardware and system model characteristics are reflected in performance predictions and can be varied easily, enabling an exploration of the parameter space. In [15] and [26], toolkits for performance predictions of high-performance parallel scientific codes are presented. While closely related to our work methodically, the authors focus on highly detailed modeling of the hardware and do not share our goal of a more generalized comprehension of performance potentials of computer network models which is reflected by SONSim's more abstract hardware model. Ewald et al. [11] presented another approach to performance prediction of parallel simulations, not relying on traces. Their goal, however, is the performance evaluation of parallel simulators and of the synchronization algorithms used instead of investigating model characteristics relevant to parallel per-

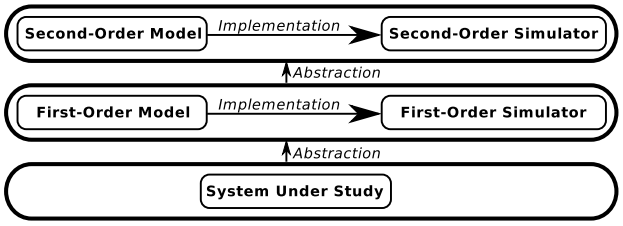

Figure 1: Levels of abstraction in modeling of simulations.

formance. Existing works with intents and abstraction levels closer to ours seem to lack a thorough evaluation for multiple simulation models and hardware environments. While Swope et al. [30] give no evaluation of prediction accuracy, both Perumalla et al. [21] and Juhasz et al. [17] compare performance predictions to physical simulator performance for a single specific simulation model and hardware platform. We evaluate SONSim's predictions with reference to measurements made for three fundamentally differing distributed network simulations running on physical hardware linked by two modern and commonly used interconnects: Ethernet and InfiniBand. With respect to modeling detail and intent, SONSim lies between performance models with coarse abstractions such as critical path analysis and highly detailed simulative models.

In summary, our performance model *considers the simulation execution hardware more closely than analytical methods*, yet in contrast to related work using measurements or simulative approaches employing detailed hardware models *is not tied to specific simulator implementations or low-level hardware specifics*. Therefore, we argue that SONSim enables reasonably accurate predictions, but at the same time is abstract enough to allow us to focus on identifying fundamental performance characteristics arising from properties of the investigated network simulation models.

3. PERFORMANCE PREDICTION METHODOLOGY

In this section, we describe the methodology used by SONSim to obtain performance predictions. Based solely on information gathered from an existing sequential simulator and simple network measurements, SONSim predicts the performance of a distributed implementation of the simulator, enabling decisions on whether parallelization of a simulation will give a performance benefit. In addition, we discuss the effort required to apply our prototypical implementation to existing simulators.

3.1 Levels of Abstraction

Figure 1 illustrates the levels of abstraction involved in simulating a distributed simulation. The original existing or envisioned system under study (e.g. a network of computers) is abstracted from by means of a *first-order model*. The executable implementation of the first-order model is the *first-order simulator* whose performance we intend to investigate. To this end, a model of the state and behavior of the first-order simulator is created. The resulting *second-order model* is implemented in a corresponding *second-order simulator*.

As we are concerned with network simulation, for us the original system under study is a network of computers which we will refer to as the ***network under study***. We contrast

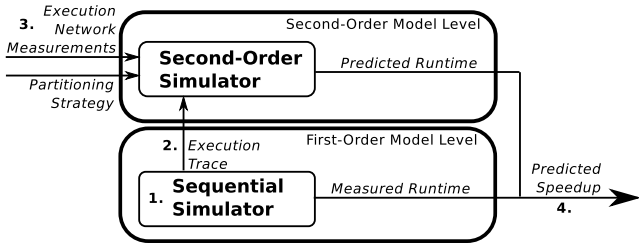

Figure 2: Data flow during performance prediction.

the ***network under study*** with the network a distributed first-order simulator is executed in, which in the following we will refer to as the ***execution network***.

3.2 Prediction Process

We will now describe the concrete sequence of activities performed to obtain performance predictions (cf. Figure 2).

1. The existing sequential first-order simulator is executed multiple times for a given scenario to determine the average sequential runtime as a reference for speedup calculation.
2. The sequential first-order simulator is instrumented to perform time measurements for execution of individual event types and to generate an *execution trace*. The execution trace contains the sequence of event executions including event timestamps and the mapping of individual events to nodes of the network under study in the first-order simulation, as well as the sequence of events being created during simulation. The instrumented sequential simulator is executed to obtain the execution trace. Note that the sequential run in step 1. is performed without tracing to avoid overheads.
3. As a basis for predicting the network overhead involved in the distributed simulation, we measure the average time required for individual message transfers in the execution network.
4. Finally, after supplying the execution network measurements, a partitioning of the first-order network model and the execution trace to the SONSim implementation, a performance prediction for the distributed first-order simulation is obtained.

3.3 Deployment

Our approach can be applied to existing discrete-event simulators easily if the simulator source code is available. Step 2. of the prediction process (cf. 3.2) requires instrumentation of the event handling code in the simulator with simple timing calls. In a discrete-event simulator, event handlers are usually implemented as a separate procedure per event type. Hence, the placement of timing calls poses no substantial difficulty. All subsequent steps are simulator-agnostic. Our prototypical implementation of SONSim with the code used for validation is available from our web site[1].

3.4 First-Order and Second-Order Model

In this section, we describe the entities constituting the first-order and second-order models used by SONSim. Both first-order and second-order model are based on the familiar discrete-event modeling paradigm. Discrete-event models are composed of two types of entities: system objects

[1]http://dsn.tm.kit.edu/english/3367.php

describing the system state and timestamped events modeling state changes at set points in simulated time [4]. In a distributed simulation, system objects are distributed to multiple interconnected machines. The simulator instances handling a subset of all system objects are called *logical processes* (LPs). Consistent with this modeling approach, networks under study are commonly reflected by first-order models as follows.

- **System objects** represent the nodes in the network under study.
- **Events** model transmissions and receptions of packets by individual nodes.
- **Logical processes** run on nodes of the execution network, each storing a number of system objects and executing events pertaining to these system objects.

Applying the same modeling pattern again, we represent state and behavior of the distributed simulation in a sequential *second-order* simulation as follows.

- **System objects** represent the logical processes of the first-order simulation.
- **Events** model activities performed by individual logical processes in the distributed simulation: execution network operations and execution of first-order events.
- As the second-order simulator itself is executed sequentially, only a single **logical process** is executed on the physical hardware.

To be able to predict the runtime of first-order simulations, a model of the operations executed by each logical process of a first-order simulator is required. A sequential discrete-event simulator operates in a simple loop: all events to be executed are stored in a queue. In each step, the event with the lowest timestamp is executed and removed from the queue. If the execution of the event triggers the creation of further events, the newly created events are added to the queue. With the addition of the execution network operations, we model each logical process' behavior as illustrated by the execution loop listed in pseudo code in Algorithm 1.

Algorithm 1 Model of the execution of each logical process as a basis for runtime prediction.

```
repeat
  repeat
    if event incoming then
      receive event
      enqueue event
    end if
  until no event received
  execute next event
  if new events created then
    enqueue locally or transfer to remote logical process
  end if
until no events left
```

The distributed execution loop extends the basic sequential discrete-event logic in two ways: first, before proceeding with execution of the next local event, logical processes probe for incoming events sent by other logical processes. Second, if a newly created event pertains to a system object handled by a remote logical process, the event is transferred to the remote logical process. Reception, execution and transfer are each associated with a runtime cost, which is added to simulation runtime for the corresponding logical process. The sum of these costs corresponds to the predicted runtime for the logical process. The maximum of the predicted runtimes among all logical processes is SONSim's prediction for the total distributed simulation runtime.

3.5 Second-Order Simulator Operation

In this section, we describe SONSim's operation, detailing the mapping of first-order to second-order simulation events. When supplied with an execution trace of a first-order simulation, measurements of the execution network and a partitioning of the model, the second-order simulator loads the execution trace and translates all initial first-order events to second-order events. There are three types of second-order events, corresponding to the operations performed in the first-order simulator execution loop: *Execution* of an event by a logical process, *Transfer* of an event to another logical process and *Reception* of an event via the execution network. SONSim operates by executing second-order events in timestamp order until no events are left in the queue. Each second-order event extends the position in simulated time according to the time measurements used as input to SONSim. On termination of the second-order simulation, the final position in simulated time represents SONSim's prediction of the first-order simulation runtime. In the second-order simulation, the second-order events are associated with the following operations.

- **Execution**: simulate creation of additional first-order events, if triggered by the event modeled. If a newly created event is associated with a system object handled by a remote logical process, a second-order *Transfer* event is scheduled for the remote first-order logical process.
- **Transfer**: respecting the target first-order logical process' state, find the arrival time of the first-order event at the target logical process and schedule a second-order *Reception* event at this point in simulated time.
- **Reception**: schedule an *Execution* event for the first-order logical process associated with the *Reception* event, respecting the remaining time required for all *Reception* and *Execution* events already scheduled.

All idle times between second-order events are considered to be spent on probing for incoming messages.

Since each logical process can receive events with arbitrary timestamps from other logical processes, explicit synchronization using an appropriate algorithm is required to maintain timestamp order. A common synchronization algorithm for parallel and distributed discrete-event simulations is the Chandy-Misra-Bryant algorithm [8, 9]. The algorithm is based on the notion of the *lookahead*, which we will use to denote the distance in simulated time from the last executed event of a logical process to the timestamp of the earliest new event that may be transferred to a remote logical process. The lookahead can be used to identify which events can be executed safely without violating timestamp order. After a second-order simulation run, SONSim returns two pieces of information: First, the *runtime of the envisioned distributed simulation* using the configured execution network hardware, simulation model and partitioning. Comparing this value to the measured sequential runtime, we can predict the benefit of parallelization. Second, a *distributed simulation schedule* allowing for examination of the first-order logical processes' interactions during execution.

3.6 Step-by-Step Analysis

In this section, we discuss the aspects of the performance of distributed simulations we will focus on in turn, motivating which performance-related characteristics need to be investigated in each of our experiments. The most important questions about a simulation model that can be answered using SONSim can be summarized as follows: **1.** Does the simulation model in itself exhibit a sufficient level of parallelism? **2.** How well must the parallelism in the model be understood at simulation runtime to achieve a speedup? **3.** What is an efficient simulation scale with respect to the number of logical processes to distribute the simulation to?

To address question 1., we need to determine upper performance bounds enabled by the parallelism in the simulation model without considering specific synchronization algorithms. Therefore, we will first assume infinite lookahead. As in [2], for experiments with infinite lookahead, synchronization is achieved by providing knowledge of all events in the simulation to all logical processes based on an execution trace created by the sequential first-order simulator, allowing logical processes to execute events as soon as possible without violating timestamp order per system object. In Section 6.2, we will proceed to demonstrate how SONSim is used to answer question 2. by investigating the performance of simulation models with synchronization under various amounts of lookahead. We demonstrate how to address simulation scale (question 3.) in Section 6.3.

4. EXPERIMENTS

The accuracy of the predictions made using SONSim was evaluated by comparing performance predictions to measurements of distributed simulation runs on physical hardware. We implemented three network models in a basic discrete-event simulation engine built from scratch, supporting sequential and distributed runs in an execution network.

The evaluation process is comprised of the following steps (cf. Section 3.2): **1.** We measure the *sequential runtime* of a simulation executed on physical hardware. **2.** An *execution trace* is gathered during a sequential simulation run on physical hardware. **3.** The time required for individual network operations is measured in the execution network. **4.** Supplied with the information from the previous steps, SONSim predicts the distributed performance. Finally, we supply the execution trace to the distributed implementation of the simulator created for evaluation to obtain a measurement of the simulation performance on physical hardware in the execution network. By comparing predicted and measured simulation performance, the prediction accuracy is evaluated.

We perform evaluation in two different execution networks, using up to six nodes for simulation: first, a cluster equipped with Intel Xeon E5504 processors operating at 2.00GHz interconnected using GBit/s Ethernet. Communication is performed using TCP. Second, a high performance computing cluster equipped with Intel Xeon E5540 processors at 2.53GHz interconnected using InfiniBand 4X QDR.

4.1 Considered Network Models

We base our experiments on three specific models of networks under study. The models are characterized by three properties: the simulated network topologies, the communication patterns between nodes and the level of detail in the modeling of the networking stack. *Packet-level* network models use packets as the smallest unit of consideration, modeling packet loss using probabilistic methods. In contrast, *signal-level* network models accurately consider effects of the physical channel on packet transmissions. In general, more detailed modeling of the networking stack translates to events that are more computationally intensive and associated with larger amounts of data. As even for small simulations, it is not clear to what extent specific types of network models benefit from parallelization, we limit our experiments in this work to distributed simulations using up to six logical processes simulating up to 1000 nodes.

Of course, the topologies of the considered networks under study display high degrees of symmetry with obvious partitionings of the models to logical processes. For more asymmetrical topologies and communication patterns, it is necessary to seek workload distributions tailored to the given topology and communication patterns, which for large simulation is a non-trivial problem. While out of scope for this paper, SONSim can already be used to guide efficient partitioning of models by repeating performance predictions for multiple candidate partitionings generated manually or using existing graph partitioning toolkits.

The following three network models were implemented in the first-order simulator to be run on physical hardware sequentially and in parallel.

1. **Interconnected Campus Networks:** As a baseline with respect to event sizes and computational demands, we study a packet-level model with a simple topology composed of a set of so-called campus networks, which can be used as building blocks to construct larger and more complex topologies [12]. A campus network is comprised of a set of nodes, each connected to a shared router. Larger networks are constructed by interconnecting multiple campus network routers. The communication patterns in the network can be influenced by configuring the amount of traffic *between* campus networks (*cross-traffic*) in relation to traffic *within* campus networks.

2. **Peer-to-Peer Networks:** Models of peer-to-peer networks are regarded as benefitting particularly little from parallel simulation [22]. We base our investigation of peer-to-peer models on the packet-level model used for the interconnected campus networks. However, in contrast to the clearly defined communication patterns given by the structure of the campus network topology, we simulate a worst case scenario with respect to simulation partitioning: as must be assumed by any parallel simulator without far-reaching knowledge of the peer-to-peer protocol and simulation model in use, a node can send a packet to any other node at arbitrary points in simulated time. The scenario is close in behavior to simulations of peer-to-peer networks with low-traffic node interactions such as distributed hash tables.

3. **Wireless Networks:** The substantial computational demands induced by the signal processing required to emulate the physical layer render signal-level wireless network simulation an obvious target for efforts to reduce runtime through parallelization [1]. Due to the demanding signal processing required, signal-level models can be considered extreme cases with respect to event data sizes and processing times in network simulations. In contrast to the previous network models, the broadcast nature of the wireless channel requires all nodes in the proximity of a sender to process each transmitted frame. For our experiments, we extracted the signal processing procedures from PhySimWifi [20], an

Table 1: Execution network measurements and event execution times from sequential simulation runs with 95% confidence intervals for the three networks under study.

Operation	Ethernet			InfiniBand		
	Campus	Peer-to-Peer	Wireless	Campus	Peer-to-Peer	Wireless
MPI Transfer	*fit function*		*fit function*	1.72±0.05μs		85.08±4.11μs
MPI Reception	0.36±0.01μs		12.97±0.12μs	0.39±0.01μs		72.07±3.53μs
MPI Probing	0.61±0.02μs		0.60±0.00μs	0.37±0.00μs		0.39±0.03μs
Send Event	3.08±0.00μs	3.26±0.02μs	997.25±13.87μs	3.07±0.03μs	3.25±0.02μs	933.17±64.60μs
Receive Event	0.91±0.00μs	0.87±0.01μs	8085.21±7.67μs	1.07±0.01μs	1.09±0.03μs	12291.92±538.51μs

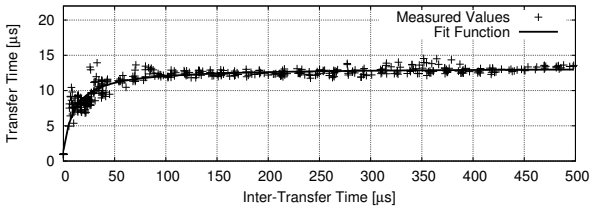

Figure 3: Time used per MPI transfer operation for 24 bytes of data depending on inter-transfer time in the Ethernet environment.

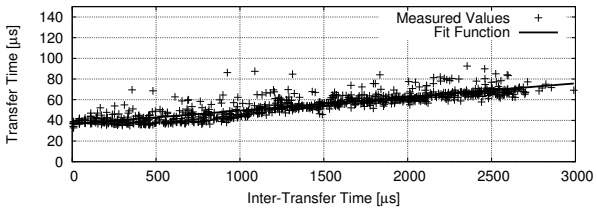

Figure 4: Time used per MPI transfer operation for 51204 bytes of data depending on inter-transfer time in the Ethernet environment.

IEEE 802.11 physical layer extension to the popular network simulator NS-3 [25].

4.2 Measuring Execution Network Overhead

For communication within the execution network, our distributed simulator implementation uses the Message Passing Interface (MPI) [28]. In the Ethernet environment, Open-MPI 1.4.1 was used, the InfiniBand experiments used HP MPI 02.03.01.00.

In order to measure the time required for MPI calls in the distributed simulator (MPI_Send, MPI_Recv, MPI_Iprobe), we implemented a simple MPI benchmark. The benchmark transfers a number of messages between a single sender and receiver, measuring the time used for each library call. Table 1 lists the time required for individual reception and probing operations for the packet-level and signal-level models with 24 and 51204 bytes of data per event, respectively.

In the Ethernet environment, we observed a substantial dependence of MPI transfer operation times on the delay between individual transfers between a given sender and receiver pair (*inter-transfer time*). To generate a basic model for the time required for transfer operations in our SONSim prototype, we applied simple curve fitting between inter-transfer times and MPI transfer operation durations in the Ethernet environment. Figures 3 and 4 depict the transfer times measured for the packet-level and signal-level models, varying inter-transfer times and the fit functions approximating the relationship between the two metrics. Given the inter-transfer time x, we get a close fit to the transfer time per message for the packet-level models using a function of the form $a/bx + c$ with $a = -131.235$, $b = 11.1845$ and $c = 13.2387$. For the signal-level model we approximate the transfer times using the fit function $mx + c$ with $m = 0.0131651$ and $c = 36.2832$. SONSim is supplied with the fit functions to predict execution network overheads depending on the inter-transfer time for each individual transfer operation. In the InfiniBand environment, transfer times did not vary significantly depending on inter-transfer times, allowing us to use averaged measured transfer times.

Table 2: Deviation of speedup predictions from measurements for the campus network (CN), peer-to-peer (P2P) and wireless network (WN) models.

	Ethernet			InfiniBand		
	CN	P2P	WN	CN	P2P	WN
Min.	3.2%	12.8%	18.3%	10.7%	34.8%	15.0%
Max.	17.1%	24.8%	19.6%	38.6%	38.6%	18.9%
Avg.	9.5%	18.0%	19.0%	21.5%	37.3%	16.3%

5. RESULTS AND DISCUSSION

We contrast the predicted speedup of distributed simulations compared to sequential runs with the speedup for distributed simulations performed in the execution networks. Table 2 lists the deviations between predictions and measured runtimes averaging over ten runs for each configuration. 95%-confidence intervals corresponded to 1.72% of total runtime on average.

Figures 5 and 6 depict the prediction and measurement results from physical simulation runs in the Ethernet and InfiniBand environments of the campus network model for two up to six logical processes. As distributed performance for the campus network model depends strongly on the proportion of cross-traffic between campus networks, we choose two levels: 10% and 50%. In the Ethernet environment, there is a close match between predicted speedup and measurement results, with an average deviation of 9.5% and a maximum deviation of 17.1% over all parameterizations. In the InfiniBand environment, we observed an average deviation of 21.5% and a maximum deviation of 38.6%. We measured a speedup factor of up to 3.18 using Ethernet and 4.27 for the InfiniBand interconnect. Given the predicted speedups of 3.68 and 4.78, a user would likely, and correctly, decide upon parallelization of the simulation.

For the peer-to-peer model (cf. Figure 7), we observed average prediction errors of 18.0% and 37.3%, respectively. The maximum deviation in the Ethernet environment was 24.8%, compared to 38.6% for InfiniBand. Parallel simulations using the InfiniBand interconnect achieve speedups of up to 2.55 compared to 1.16 for Ethernet, suggesting that parallelization in the InfiniBand environment would give some benefit. In the Ethernet environment, however, dis-

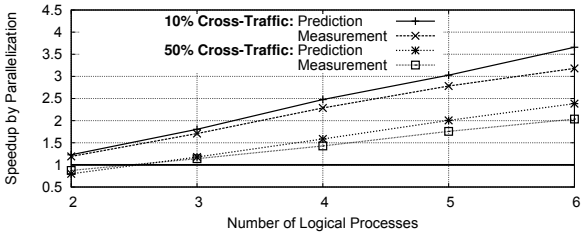

Figure 5: Evaluation of prediction accuracy for the campus network model depending on the number of logical processes and campus networks for two levels of cross-traffic in the Ethernet environment.

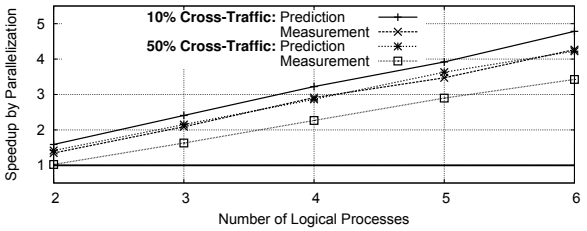

Figure 6: Evaluation of prediction accuracy for the campus network model depending on the number of logical processes and campus networks for two levels of cross-traffic in the InfiniBand environment.

tributed simulation with up to six logical processes is barely worthwhile, even assuming infinite lookahead. Based on speedup predictions of 3.51 and 1.44, we assume that a user of SONSim would come to the same conclusions.

In contrast to the results for the campus network and peer-to-peer network models, prediction accuracy for distributed simulations of the wireless network model (cf. Figure 8) is comparable between Ethernet and InfiniBand interconnects. We observed average deviations of 19.0% and 16.3%, and maximum deviations of 19.6% and 18.9%, respectively. The higher prediction accuracy compared for the wireless model is explained by its high computational demands. As network overheads make up a substantially lower proportion of the runtime, the impact of inaccuracies in the prediction of network overheads is less pronounced. Measured speedups are slightly lower for Ethernet, with a maximum speedup of 4.25 compared to 4.77 for InfiniBand. Again, given predicted speedups of 5.06 and 5.67, a user would likely decide upon parallelization and achieve performance gains as expected.

Considering all experiments, we observe that predictions are consistently overly optimistic. The limitation in prediction accuracy is caused by an underestimation of execution network overheads by SONSim which reduces predicted runtime. As increasing the number of logical processes reduces the event execution time for each logical process, inaccuracies in the predictions for the network overhead gain in impact. Therefore, for larger numbers of logical processes, final prediction accuracy decreases. In future work, we will integrate an existing and more detailed execution network model in SONSim, which we hope will further improve prediction accuracy.

As a basis for discussing the sufficiency of SONSim's prediction accuracy, we compare our results to the related work discussed in Section 2 with comparable intent and level of modeling detail to SONSim. Juhasz et al. [17] evaluated their performance prediction tool with respect to a single unspecified simulation model and in a single execution network,

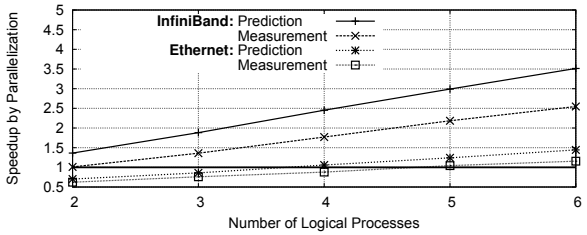

Figure 7: Evaluation of prediction accuracy for the peer-to-peer network model depending on the number of logical processes.

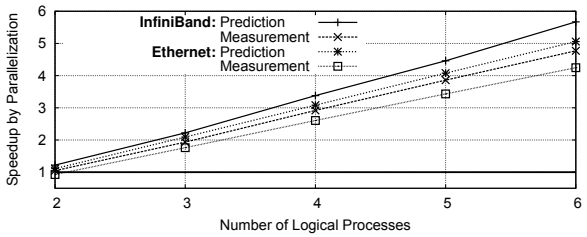

Figure 8: Evaluation of prediction accuracy for the wireless network model depending on the numbers of logical processes and nodes.

noting that for low event processing times, diminishing prediction accuracy was observed. Prediction errors were below 15% for five or less logical processes, with deviations of up to 66% for six and seven logical processes. In [21], Perumalla et al. evaluate performance predictions for a physics simulation model in a single execution network, achieving prediction errors of about 10% to 20%. Perumalla et al. point out that for the purpose of performance predictions, some deviation can be tolerated. As substantial performance gains are required to justify the implementation effort for parallelization of a model, even SONSim's maximum observed prediction error of 38.6% in the InfiniBand environment might still be acceptable for making informed decisions on parallelization. Hence, we consider SONSim's prediction accuracy an acceptable starting point for its intended purpose. Nevertheless, especially with regard to large execution networks, integrating an established and more accurate model of the execution network is a focus of our ongoing work.

6. UNDERSTANDING SPEEDUP POTENTIALS

Having evaluated strengths and limitations of SONSim's predictions, we demonstrate the performance statements that can be made using SONSim. As model and hardware parameters can be varied easily in the second-order simulation, general statements about a network models potential for parallelization depending on the parameters can be made. We show this usage of SONSim by varying the degree of modeling detail, the lookahead and the simulation scale and examine the effects on distributed simulation performance.

6.1 Impact of Modeling Detail

As a first example, based on the observation that higher levels of modeling detail are typically reflected by more computationally intensive events, we vary the execution time per event for the three reference network models to investigate the impact of the level of detail in the model of the network under study on the distributed simulation performance.

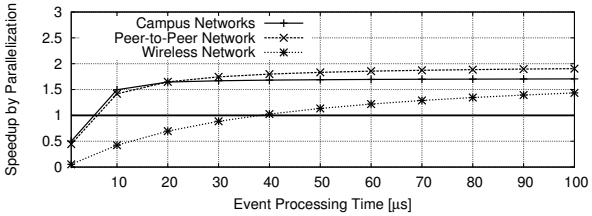

Figure 9: Predictions for the speedup by distributed simulation of the three network models using two logical processes in the Ethernet environment, varying processing times per event.

Figure 9 shows SONSim's speedup predictions for distributed simulations using two logical processes in the Ethernet environment. For the campus network model with 50% of cross-traffic, a performance gain is achieved for events associated with processing times larger than about 4μs, not adjusting for SONSim's current overestimation of speedups.

Similar to the predictions for the campus network model, a modest increase in processing time for each event suffices to make distributed simulation of the peer-to-peer model worthwhile even for only two logical processes. This indicates that with sufficient lookahead, distributed simulation of peer-to-peer networks could have a substantial benefit. However, achieving large lookahead can be difficult depending on the network protocols modeled. In 6.2, we will demonstrate how SONSim can be used to examine how much lookahead is required for specific types of network models.

Even for the wireless model, where events are associated with 51204 bytes of data each compared to 24 bytes for the packet-level models, considerable speedup is achieved for comparatively low processing times per event. This is remarkable considering that for every frame transmitted in the wireless network under study, an event is transferred in the execution network.

Based on the prediction results, we note the clear difference in event processing times required for performance gains through parallelization between the packet-level and signal-level network models and identify event processing times as an obvious influencing factor to distributed performance. In our future work, we intend to conduct a systematic study of the influencing factors to establish a classification of computer network models according to their parallelization potential.

6.2 Impact of Model Lookahead

It is well-known that distributed simulation performance is highly dependent on the amount of lookahead available in the given model. As discussed briefly in Section 3.5, lookahead describes the amount of simulated time for which a logical process will not send any new events to remote logical processes, which for all other logical processes is closely related to the amount of simulated time for which it is safe to execute events from the local queue. The upper bound for the amount of lookahead that can be extracted from a network model depends on properties such as the topology of the simulated network, link latencies, communication patterns and the considered network protocols. Some amount of lookahead can be extracted statically based on knowledge of the model of the network under study, e.g. using the known minimum message latency between nodes simulated on different machines of the execution network. If more lookahead is required for reasonable performance, the simulator can dynamically calculate the lookahead depending on the state of the simulated system at a given point in time. This calculation may be quite expensive depending on how deeply the system state must be examined. Hence, dynamic calculation of the lookahead carries a tradeoff between the amount of processing time spent on lookahead maximization and the performance gains resulting from larger lookahead values. Besides guiding decisions on how much computational and developmental effort should be expended on determining the maximum lookahead value, SONSim can help answering the more general question *"How much lookahead is required for a distributed simulation of the given model to perform well?"*.

In our description of the lookahead experiments we loosely follow the terminology of Bagrodia et al. [2].

- *Lookahead*: the lowest possible delta between the timestamp of the event currently executed by a logical process and the timestamp of any locally created event to be executed on another logical process.
- *Earliest Conditional Output Time* (*ECOT*): for each logical process, the ECOT is the earliest possible timestamp of a locally created event to be executed on another logical process.
- *Earliest Input Time* (*EIT*): for each logical process, the EIT is the earliest possible timestamp of an event which will be received from another logical process. A lower bound for the EIT can be determined from the minimum of all other logical processes' ECOTs.

To perform lookahead experiments, SONSim's behavior was extended as follows.

- After a logical process has executed an event, the logical process determines its ECOT and stores it in a variable accessible to all other logical processes. For the example presented here, the ECOT is chosen as the sum of the timestamp of the event executed in this step and a fixed lookahead value.
- Before executing an event, a logical process determines the EIT, which is the minimum of all remote ECOTs. All events with timestamps lower than the minimum ECOT are considered safe to execute. If no such event exists, the logical process waits.
- If all logical processes are waiting for the EIT to advance beyond the timestamp of any of the events in the local queue, the simulation is deadlocked. Deadlocks are resolved using a simple policy: all logical processes advance their ECOT to the timestamp of the earliest event in the queue, allowing the simulation to proceed.

In effect, SONSim's behavior simulates an idealized version of the Chandy-Misra-Bryant protocol disregarding overheads for signaling of ECOTs. We can now study the effects of various quantities of lookahead in given network models by varying the lookahead and predicting the performance of the resulting distributed simulations.

Figure 10 shows the predicted performance of distributed simulations for the campus network model in the Ethernet execution network using two logical processes. We investigated the performance for setups with very low lookahead (5μs) up to very large lookahead (1s). The predicted speedup by distributed simulation varies from 0.961 to 1.163 for 10% of cross-traffic and from 0.469 to 0.689 for 50% of

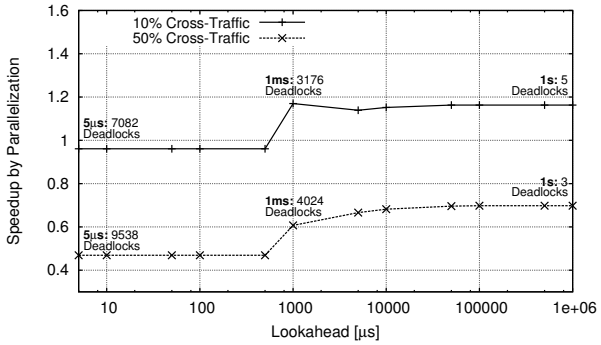

Figure 10: Predictions for the speedup by distributed simulation of the three network models using two logical processes in the Ethernet environment, varying the amount of lookahead.

cross-traffic. For 1s of lookahead, the predicted speedups closely approach the results measured under the assumption of infinite lookahead in our validation (cf. Figure 5). There is no discernible difference in the speedup between 5 and 500μs of lookahead. The reason is that for these low amounts of lookahead, the simulation advances almost solely due to deadlock resolution. The large increase in speedup for 1000μs of lookahead is explained by the spacing of 1000μs between initial send events in our example simulation model. As in many cases, the earliest next event to be executed by a logical process lies 1000 μs in the future, lower lookahead values tend to not allow us to be sure about the next event being safe to execute, increasing the probability for waiting and for deadlocks. Nevertheless, as there is no runtime penalty for deadlock resolution in this experiment, speedup values remain relatively large even for large numbers of deadlocks. As can be seen by comparing the speedup for 1000 μs and 5000μs of lookahead and 10% of cross-traffic, larger numbers of deadlocks can actually be beneficial to the predicted speedup if there is no overhead involved, as during deadlock resolution, ECOTs are increased for all logical processes. As an example, if we were to introduce a penalty of 10 μs for deadlock detection and resolution, the predicted speedup for 5μs of lookahead would drop to 0.367 for 10% of cross-traffic and to 0.256 for 50% of cross-traffic.

In summary, we note that – as expected – speedup predictions vary substantially with varying amounts of lookahead, especially when factoring in overheads for deadlock resolution. Of course, the largest investigated lookahead value of 1s cannot be expected for most real-world simulations without deep insight into the simulation model during runtime, and gaining this insight might be associated with high computational demands in itself. However, knowledge about what level of performance is to be expected for large lookahead values can guide decisions on how much computational and developmental effort should be expended on dynamic lookahead maximization. Additionally, when investigating classes of network models, the results might show that performance would require unattainably large amounts of lookahead, rendering parallelization of these classes of networks models a futile endeavor.

6.3 Determining Simulation Scale

Given a simulation model which is known to lend itself to parallelization, an appropriate simulation scale has to be selected, as simulation performance may stagnate or degrade

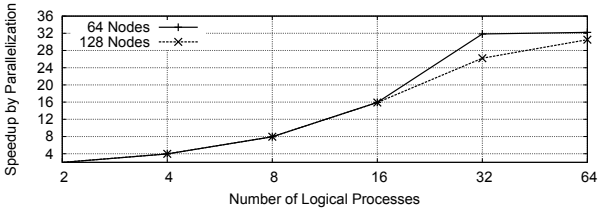

Figure 11: Predictions for the speedup by distributed simulation of the wireless network model in the Ethernet environment, varying the number of logical processes.

beyond a certain number of logical processes. Particularly considering today's highly scalable execution environments such as cloud services employing usage-based billing, maximizing efficiency becomes an important issue. By varying the number of logical processes in a simulation, we demonstrate how SONSim can help determining scalability limits of network models for given hardware environments. As an example, Figure 11 shows the predicted speedup by parallelization of simulations of the wireless network model in the Ethernet environment for simulations of 64 and 128 nodes. As we can see, for 64 nodes, predicted speedups scale close to linearly for up to 32 logical processes, with a predicted speedup of 31.85. Beyond 32 logical processes, diminishing returns are observed due to the increasing influence of execution network overheads. As the speedup for 64 logical processes is only 32.21, more than 32 logical processes are not useful in this scenario. For simulations of 128 nodes, performance scales slightly worse, with a speedup of 26.20 for 32 logical processes. Using 64 logical processes, the predicted speedup increases only slightly to 30.52. As the predictions are based on the basic model of the execution network implemented in the SONSim prototype, we assume the results shown to only give rough estimates for large and congested execution networks. Reliable prediction of optimal simulation scale will be addressed based on the more detailed modeling of the execution network overheads as part of our future work, which will allow for simulation scale to be included in our model classification.

7. CONCLUSION

We presented SONSim, an approach towards performance evaluation of distributed simulations of computer networks. SONSim predicts the performance of distributed discrete-event network simulations to determine whether parallelization of an existing sequential simulator is worth the required development effort. In addition, by varying model parameters, SONSim's predictions can help identifying the network model and hardware characteristics limiting simulation performance. We demonstrated this usage by varying the computational intensity, the available lookahead and the simulation scale for three models and predicting the effect on distributed performance. The prediction accuracy of our prototypical implementation was evaluated for three common network topologies simulated on two hardware platforms.

In our future work, we will extend and improve our prototypical implementation, in particular with respect to performance prediction accuracy. We expect that building upon an existing accurate model of the execution network will result in closer performance estimates for large simulations. Our intention is to utilize SONSim to get a clear under-

standing on which conditions are required for high performance in distributed simulations of various simulated network types and considering characteristics of existing model implementations in common simulators. Ideally, the gained understanding will enable us to establish a classification of computer network models with respect to their potential for parallelization.

8. ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their insights and helpful comments.

9. REFERENCES

[1] P. Andelfinger, J. Mittag, and H. Hartenstein. GPU-Based Architectures and Their Benefit for Accurate and Efficient Wireless Network Simulations. In *19th Int'l Symp. on Modeling, Analysis and Simulation of Comp. and Telecomm. Systems*, pages 421–424, 2011.

[2] R. Bagrodia and M. Takai. Performance Evaluation of Conservative Algorithms in Parallel Simulation Languages. *IEEE Transactions on Parallel and Distributed Systems*, pages 395–411, 2000.

[3] R. L. Bagrodia. Perils and Pitfalls of Parallel Discrete-Event Simulation. In *Proc. of the 28th Winter Simulation Conference*, pages 136–143. IEEE Computer Society, 1996.

[4] O. Balci. The Implementation of Four Conceptual Frameworks for Simulation Modeling in High-Level Languages. In *Proceedings of the Winter Simulation Conference*, pages 287–295, 1988.

[5] L. Barriga, R. Ronngren, and R. Ayani. Benchmarking Parallel Simulation Algorithms. In *Proceedings of the IEEE First International Conference on Algorithms and Architectures for Parallel Processing*, 1995.

[6] S. Becker, H. Koziolek, and R. Reussner. Model-Based Performance Prediction with the Palladio Component Model. In *Proceedings of the 6th International Workshop on Software and Performance*, pages 54–65. ACM, 2007.

[7] O. Berry and D. Jefferson. Critical Path Analysis of Distributed Simulation. In *Proceedings of the 1985 SCS Multiconference on Distributed Simulation*.

[8] R. E. Bryant. Simulation of Packet Communication Architecture Computer Systems. Technical report, 1977.

[9] K. Chandy and J. Misra. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering*, SE-5(5):440–452, 1979.

[10] S. De Munck, K. Vanmechelen, and J. Broeckhove. Revisiting Conservative Time Synchronization Protocols in Parallel and Distributed Simulation. *Concurrency and Computation: Practice and Experience*, 2013.

[11] R. Ewald, J. Himmelspach, A. Uhrmacher, D. Chen, and G. Theodoropoulos. A Simulation Approach to Facilitate Parallel and Distributed Discrete-Event Simulator Development. In *IEEE Int'l Symposium on Distr. Simulation and Real-Time Applications*, pages 209–218, 2006.

[12] R. Fujimoto, K. Perumalla, A. Park, H. Wu, M. Ammar, and G. Riley. Large-Scale Network Simulation: How Big? How Fast? In *11th IEEE/ACM Int'l Symposium on Modeling, Analysis and Simulation of Computer Telecommun. Systems*, pages 116–123, 2003.

[13] D. Gianni, G. Iazeolla, and A. D'Ambrogio. A Methodology to Predict the Performance of Distributed Simulations. In *Workshop on Principles of Advanced and Distributed Simulation (PADS)*, pages 31–39, 2010.

[14] A. Gupta, L. V. Kalé, D. S. Milojicic, P. Faraboschi, R. Kaufmann, V. March, F. Gioachin, C. H. Suen, and B.-S. Lee. Exploring the Performance and Mapping of HPC Applications to Platforms in the Cloud. In *Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing*, HPDC, pages 121–122, 2012.

[15] S. D. Hammond, G. R. Mudalige, J. A. Smith, S. A. Jarvis, J. A. Herdman, and A. Vadgama. WARPP: a Toolkit for Simulating High-Performance Parallel Scientific Codes. In *Proc. of the 2nd Int'l Conference on Simulation Tools and Techniques*, pages 19:1–19:10, 2009.

[16] Q. He, M. Ammar, G. Riley, H. Raj, and R. Fujimoto. Mapping Peer Behavior to Packet-Level Details: a Framework for Packet-Level Simulation of Peer-to-Peer Systems. In *11th IEEE/ACM Int'l Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems*, pages 71–78, 2003.

[17] Z. Juhasz, S. Turner, K. Kuntner, and M. Gerzson. A Performance Analyser and Prediction Tool for Parallel Discrete Event Simulation. In *UKSIM 2001: Conference On Computer Simulation*, pages 148–155, 2001.

[18] P. Konas and P.-C. Yew. Parallel Discrete Event Simulation on Shared-Memory Multiprocessors. In *Proc. of the 24th Annual Simulation Symposium*, pages 134–148, 1991.

[19] J. Liu, D. Nicol, B. Premore, and A. Poplawski. Performance Prediction of a Parallel Simulator. In *Thirteenth Workshop on Parallel and Distributed Simulation*, pages 156–164, 1999.

[20] J. Mittag, S. Papanastasiou, H. Hartenstein, and E. Ström. Enabling Accurate Cross-Layer PHY/MAC/NET Simulation Studies of Vehicular Communication Networks. *Proceedings of the IEEE*, 99(7):1311–1326, 2011.

[21] K. Perumalla, R. Fujimoto, P. Thakare, S. Pande, H. Karimabadi, Y. Omelchenko, and J. Driscoll. Performance Prediction of Large-Scale Parallel Discrete Event Models of Physical Systems. In *Proc. of the Winter Simulation Conference*, pages 356–364, 2005.

[22] M. Quinson, C. Rosa, and C. Thiery. Parallel Simulation of Peer-to-Peer Systems. In *CCGrid 2012 – The 12th IEEE/ACM Int'l Symposium on Cluster, Cloud and Grid Computing*, pages 668–675, 2012.

[23] P. F. Reynolds, Jr., and P. M. Dickens. SPECTRUM: A Parallel Simulation Testbed. In *Proceedings of the Hypercube Conference*, 1989.

[24] G. F. Riley, M. H. Ammar, R. M. Fujimoto, A. Park, K. Perumalla, and D. Xu. A Federated Approach to Distributed Network Simulation. *ACM Transactions on Modeling and Computer Simulation*, pages 116–148, 2004.

[25] G. F. Riley and T. R. Henderson. The NS-3 Network Simulator. In *Modeling and Tools for Network Simulation*, pages 15–34. Springer Berlin Heidelberg, 2010.

[26] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. Cooper-Balls, and B. Jacob. The Structural Simulation Toolkit. *SIGMETRICS Performance Evaluation Review*, 38(4):37–42, 2011.

[27] C. A. Schaefer, V. Pankratius, and W. F. Tichy. Engineering Parallel Applications with Tunable Architectures. In *Proc. of the 32nd ACM/IEEE Int'l Conference on Software Engineering - Volume 1*, pages 405–414, 2010.

[28] M. Snir, S. W. Otto, D. W. Walker, J. Dongarra, and S. Huss-Lederman. *MPI: The Complete Reference*. MIT Press, 1995.

[29] H. Sutter and J. Larus. Software and the Concurrency Revolution. *Queue*, 3(7):54–62, 2005.

[30] S. M. Swope and R. M. Fujimoto. Optimal Performance of Distributed Simulation Programs. In *Proc. of the 19th Winter Simulation Conference*, 1987.

[31] K. Vanmechelen, S. De Munck, and J. Broeckhove. Conservative Distributed Discrete Event Simulation on Amazon EC2. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGRID, pages 853–860, 2012.