Horizon – Exploiting Timing Information for Parallel Network Simulation

Georg Kunz, Olaf Landsiedel, Klaus Wehrle
Distributed Systems Group
RWTH Aachen University
{kunz,landsiedel,wehrle}@cs.rwth-aachen.de

I. Introduction

Network simulation faces an increasing demand for highly detailed simulation models which in turn require efficient handling of their inherent computational complexity. This demand for detailed models includes both accurate estimations of processing time and in-depth modeling of wireless technologies. For instance, one might want to investigate if a particular device can incorporate a computationally complex radio transmission technology while meeting the deadlines of a multi-media streaming application such as VoIP.

Current network simulators however do not incorporate the processing time of events, but merely assume that each event is processed in zero simulation time. This abstraction makes it particularly difficult to accurately conduct processing time measurements. Moreover, detailed modeling of e.g. radio technologies increases the computational complexity of simulations and hence their run-time significantly. Existing approaches to parallel simulation [1] attempt to reduce simulation run-times, but perform poorly in the context of wireless networks due to the limitations and run-time overhead of their synchronization algorithms. While classic parallel simulation focuses on computing clusters, we argue that the proliferation of multi-core computers presents a cheap and valuable alternative.

This paper presents Horizon – an extension to network simulation that enables the efficient and detailed simulation of wireless networks. Our contributions are two-fold as Horizon provides i) an API for accurately modeling processing time of discrete event simulation models by augmenting events with time spans and ii) a lightweight parallelization scheme that utilizes timing information to guide the parallel execution of simulations on multi-core computers. In this paper we primarily focus on the latter.

II. Problems of Classic Parallelization

Classic parallel discrete event simulation [1] relies on two classes of synchronization algorithms to maintain data consistency across the distributed partitions of a simulation model. Optimistic algorithms speculatively execute events in parallel and perform roll-backs once an inconsistent state is detected. Conservative algorithms in contrast avoid inconsistent states by exchanging synchronization information among the partitions. Both classes of algorithms suffer from a significant run-time overhead in particular when applied to wireless networks.

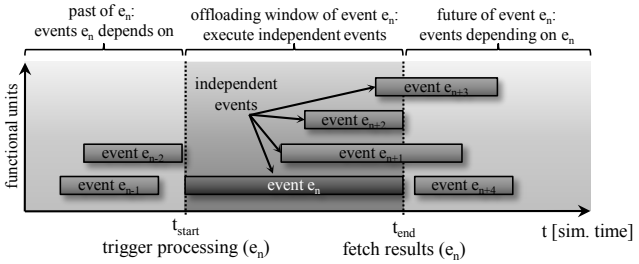

Fig. 1. The interval between t_{start} and t_{end} of event e_n opens a window for parallelization. Independent events that start in this interval can be offloaded to a different CPU.

Wireless networks, in contrast to their wired counterparts, cause significantly small lookahead values between partitions which makes it difficult to correctly predict and schedule future events. To counteract these effects, Horizon explicitly trades distributed simulation on computing clusters for efficiency and focuses on shared memory computers. Hence, Horizon offers a lightweight parallelization architecture that benefits from centralized knowledge and avoids the need for complex synchronization algorithms.

III. Horizon(tal) Parallelization

The goal of Horizon is to enable accurate modeling of processing time and achieve high run-time performance through parallelization. In particular, Horizon utilizes available timing information to guide parallel processing of independent events. This section details on Horizon's approach and its architecture.

Approach: The key idea of Horizon lies in augmenting events with an explicit duration of simulation time. Instead of occurring at a discrete point in simulation time, augmented events last from a distinct starting time (t_{begin}) to an ending time (t_{end}). In terms of processing, this approach distinguishes between the point in simulation time at which the processing of a given event is initiated and the point in simulation time at which the resulting data is needed to continue the simulation. This observation naturally opens a time window (i.e., horizon) in which the parallel processing of independent events that start in this window can be performed on different processing units. Specifically, at t_{start} the simulation kernel offloads the processing of an event to a different processor and continues handling further events until the simulation time reaches t_{end}. At t_{end}, the simulation kernel waits for the calculation to finish if needed, fetches the results, and continues (see Figure 1).

This parallelization scheme dynamically processes independent events from any layer and any node in the simulation model. Hence, we denote it *horizontal parallelization* to distinguish it from classic parallelization which partitions the simulation model vertically in terms of clusters of nodes.

Challenges: Two central challenges arise in the context of Horizon. First, horizontal parallelization must identify independent events to guarantee correctness. To achieve this, simulation models in Horizon are composed of functional units (e.g., transmitter, channel, IP). Similarly to the concept of logical processes in classic parallelization, each unit maintains private local state and interacts with neighboring functional units only via message passing [1]. As a result, events that exhibit overlapping processing times and occur in different functional units are independent. The central scheduler then guarantees two properties: First, only one event per functional unit is executed at any point in simulation time to avoid data corruption within a functional unit. Second, the global ordering of events remains intact due to the central event queue. Furthermore, to model access to shared media, Horizon provides an API for specifying that events belonging to certain functional units may never be offloaded – hereby enabling implicit synchronization points.

The second challenge regards the processing time of an event which is in general not known in advance, but may depend on run-time parameters such as the number of iterations performed during decoding a received packet. Horizon consequently offers an API for adjusting (i.e., prolonging) the (minimum) duration of an event at run-time. The scheduler directly incorporates any time adjustments by dynamically scheduling further events if possible.

Discussion: Horizon's lightweight architecture heavily relies on a central scheduler and is hence only applicable to shared-memory multi-core computers. We believe that the increasing availability of multi-core computers renders such systems a cheap and valuable alternative to full-sized computing clusters for small to medium sized simulations. However, horizontal parallelization is orthogonal to existing parallel simulation schemes. By utilizing existing parallel simulation mechanisms, Horizon transparently integrates with distributed computing clusters.

IV. Prototype and Preliminary Evaluation

We implemented a prototype based on OMNeT++ 4.0 [5] to evaluate the viability of Horizon and conduct first performance measurements. In OMNeT++, simulation models exhibit a modular structure that allows them to seamlessly integrate with the concept of functional units. Our key modifications to OMNeT++ encompass extensions to the event scheduler, a thread pool for parallel event processing, and corresponding changes to the native API.

We conducted early performance evaluations of Horizon by means of a synthetic benchmark on a 16 core (8 hyper-threaded Intel XEON 2.5GHz) computer. This benchmark models a wireless network consisting of a grid of 25 nodes which randomly send and receive packets. Sending and receiving are CPU intensive operations that are parameterized in terms of event duration to investigate its influence on performance. Figure 2 illustrates that Horizon achieves a significant speedup in comparison to typical single threaded simulators. The figure furthermore shows that the duration of an event noticeably affects performance since longer durations allow Horizon to schedule more events in parallel. Although basic, we consider these results promising.

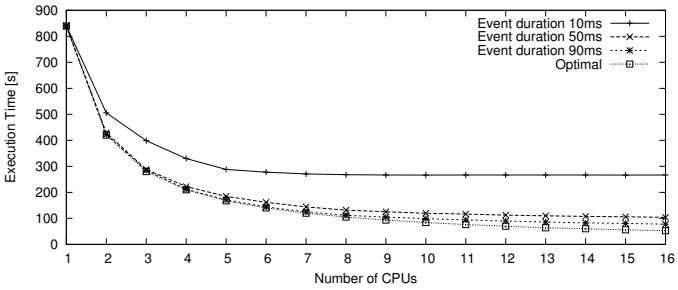

Fig. 2. Long lasting events allow Horizon to schedule more events in parallel resulting in a decreased simulation time. The graph also shows the theoretically optimal speedup.

V. Timing Calibration and Future Work

In its current incarnation, Horizon requires researchers to manually annotate events with timing information. Hence, Horizon depends on credible sources for gathering performance measurements to calibrate its own models. Such performance measurements are typically conducted by means of emulators such as Simics [3] or AVRORA [4]. Based on highly detailed system models, these tools provide accurate results at the cost of a comparatively slow run-time performance.

Future work focuses on the automation of the calibration process. Based on the presented architecture and previous work [2], our efforts target particularly the automatic derivation of performance measurements from emulators and real systems and investigate their integration to existing models.

VI. Conclusion

This paper presents Horizon, an extension to discrete event simulation to accurately and efficiently model timing behavior. Horizon achieves high run-time performance by utilizing timing information for horizontal parallelization. We additionally illustrated the viability of Horizon by means of a prototype implementation and evaluation.

Acknowledgements: This research was funded in part by the DFG Cluster of Excellence on Ultra-high Speed Information and Communication (UMIC), German Research Foundation grant DFG EXC 89.

References

[1] R. Fujimoto. Parallel discrete event simulation. In *Proceedings of Winter Simulation Conference*, pages 19–28, 1989.

[2] O. Landsiedel, H. Alizai, and K. Wehrle. When Timing Matters: Enabling Time Accurate and Scalable Simulation of Sensor Network Applications. In *Proc. of the 2008 International Conference on Information Processing in Sensor Networks (IPSN 2008)*, 2008.

[3] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A Full System Simulation Platform. *Computer*, 35(2):50–58, 2002.

[4] B. Titzer, D. Lee, and J. Palsberg. Avrora: Scalable Sensor Network Simulation with Precise Timing. In *Proc. of the 4th International Symposium on Information Processing in Sensor Networks*, 2005.

[5] A. Varga. The OMNeT++ Discrete Event Simulation System. In *Proc. of the European Simulation Multiconference (ESM)*, June 2001.