JiST: EMBEDDING SIMULATION TIME INTO A VIRTUAL MACHINE (extended abstract)

Rimon Barr, Zygmunt J. Haas, Robbert van Renesse

Computer Science and Electrical Engineering,
Cornell University, Ithaca NY 14853
{barr@cs, haas@ee, rvr@cs}.cornell.edu

ABSTRACT

We propose a new approach for constructing simulators that leverages virtual machines and combines the advantages of the traditional systems-based and language-based simulator designs. We introduce JiST, a Java-based simulation platform that executes discrete event simulations efficiently by embedding simulation semantics directly into the Java execution model and transparently performs important optimizations via bytecode-level program transformations. The system provides standard benefits that the modern Java runtime affords. In addition, JiST is efficient, out-performing existing highly optimized simulation runtimes both in space and time. We illustrate JiST's practicality by constructing SWANS, a scalable wireless network simulator.

KEYWORDS

discrete event simulation, simulation languages and environments, virtual machine, Java

1 INTRODUCTION

Due to their popularity and widespread utility, discrete event simulators have been the subject of much research (surveyed in [5, 2, 6, 3]). From a systems perspective, researchers have built many types of simulation *libraries* or execution *runtimes* spanning the gamut from the conservatively parallel to the aggressively optimistic, and from the shared memory to the message passing paradigms. From a modeling perspective, researchers have designed numerous simulation *languages* that codify event causality, execution semantics and simulation state constraints, which both simplify parallel simulation development and permit important static and dynamic optimizations. *[design objectives deleted]*

Instead, we propose a new approach to building simulators: to bring simulation semantics to a modern and popular virtual machine-based language. JiST, which stands for **J**ava **i**n **S**imulation **T**ime, is a new discrete event simulation system built along these principles, integrating the prior systems and languages approaches. Specifically, the key motivation behind JiST is to create a simulation system that can execute discrete event simulations *efficiently*, yet achieve this *transparently* within a *standard* language and its runtime, where: *[definitions deleted]*

These three attributes – the last one in particular – highlight an important distinction between JiST and previous simulation systems in that the simulation code that runs on JiST need not be written in a domain-specific language invented specifically for writing simulations, nor need it be littered with special-purpose system calls and call-backs to support runtime simulation functionality. Instead, JiST transparently introduces simulation time execution semantics to simulation programs written in plain Java and they are executed over an unmodified Java virtual machine. In other words, JiST converts a virtual machine into a simulation system that is flexible and efficient.

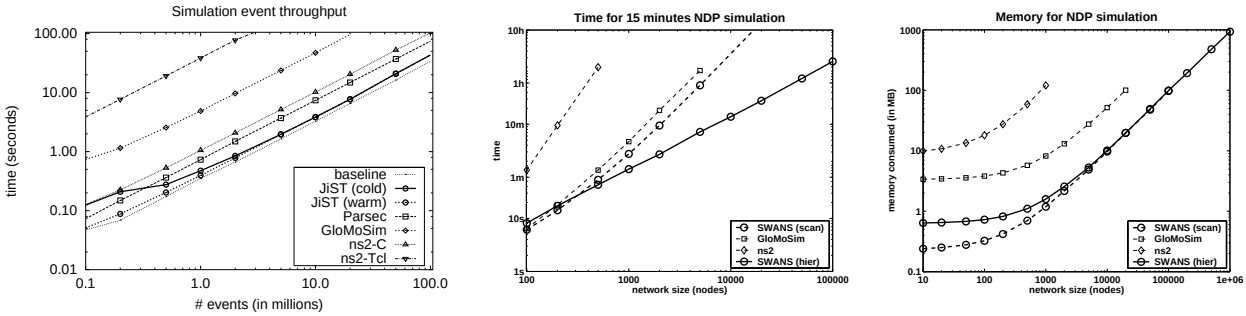
2 DESIGN AND IMPLEMENTATION

[deleted sections: **Simulation time execution**, **Simulation programs**, **System architecture**, **Rewriter**, **Simulation time kernel**, **API and semantics**, **Hello world!**, **Parallel simulation execution**.*]*

3 EVALUATION

Though conventional wisdom regarding language performance [1] might suggest against implementing our system in Java, we show that JiST and SWANS, a scalable network simulator built atop our platform, perform surprisingly well.

[experimental setup, additional results, and discussion omitted]



(a) (b) (c)

Figure 1. (a) JiST throughput micro-benchmarks. SWANS (b) throughput and (c) memory macro-benchmarks running simulations of the node discovery protocol. (Note the log-log axes.)

JiST has twice the event throughput than even the highly optimized, C-based Parsec engine, and comes within 20% of a C-based baseline comprised merely of successive heap operations! The JiST design also results in very small per entity and per event memory footprints, outperforming all the competing simulator designs. *(not shown)*

Comparing SWANS with the two most popular ad hoc wireless network simulators, ns2 [4] and GloMoSim [7], highlights the practical advantages of JiST. The results shown above are for simulations of a node discovery protocol. Within the same memory limits, SWANS can model networks that are one and two orders of magnitude larger than what is possible with GloMoSim and ns2, respectively. SWANS also has more than ten times the event throughput of ns2, and implements an efficient signal propagation protocol that allows simulation time to scale linearly with the simulated network size. The net result of the various design decisions and optimizations is a wireless network simulator that, as an example, can simulate a million node network running a node discovery protocol within approximately 6 hours on a 2.0 GHz uni-processor with 2 GB of RAM, or two orders of magnitude beyond existing network simulator capabilities.

REFERENCES

[1] D. Bagley. The great computer language shoot-out, 2001. `http://www.bagley.org/~doug/shootout/`.

[2] R. M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, Oct. 1990.

[3] R. M. Fujimoto. Parallel and distributed simulation. In *Winter Simulation Conference*, pages 118–125, Dec. 1995.

[4] S. McCanne and S. Floyd. ns (Network Simulator) at `http://www-nrg.ee.lbl.gov/ns`, 1995.

[5] J. Misra. Distributed discrete event simulation. *ACM Computing Surveys*, 18(1):39–65, Mar. 1986.

[6] D. M. Nicol and R. M. Fujimoto. Parallel simulation today. *Annals of Operations Research*, pages 249–285, Dec. 1994.

[7] X. Zeng, R. L. Bagrodia, and M. Gerla. GloMoSim: a library for parallel simulation of large-scale wireless networks. In *Workshop on Parallel and Distributed Simulation*, May 1998.