JiST – Java in Simulation Time

An efficient, unifying approach to simulation using virtual machines

Rimon Barr, Zygmunt Haas, Robbert van Renesse
rimon@acm.org haas@ece.cornell.edu rvr@cs.cornell.edu

Cornell University, USA

8 September 2004

http://jist.ece.cornell.edu/
motivation: simulation

- cost per MIPS declining
  - e.g. Pentium Xeon:
    - \( \sim 10,000 \text{ MIPS} @ \sim \$200 \)
- emphasis on computation
  - vs. analytical methods
  - vs. empirical methods
- simulators are needed
  - e.g., wireless networks
- published ad hoc network simulations
  - lack network size - \( \sim 500 \) nodes; or
  - compromise detail - packet level; or
  - curtail duration - few minutes; or
  - are of sparse density - \( <10/\text{km}^2 \)

i.e. limited simulation scalability [Riley02]

Simulation scalability is important
what is a simulation?

- **unstructured simulation**: computers compute
- **time structured**: event-oriented vs. process-oriented

**discrete event simulator** is a program that:
- encodes the simulation **model**
- stores the **state** of the simulated world
- performs **events** at discrete simulation times
- **loops** through a temporally ordered **event queue**
- works through **simulation time** as quickly as possible

**desirable properties** of a simulator:
- **correctness**: valid simulation results
- **efficiency**: performance in terms of throughput and memory
- **transparency**: separate correctness from efficiency:
  - write “simple” program in a **standard** language
  - provide implicit optimization, concurrency, distribution, portability, etc.
Virtual machine-based simulation

how do we build simulators?

systems

• simulation kernels
  • control scheduling, IPC, clock
  • processes run in virtual time
  • e.g. TimeWarp OS [Jefferson87], Warped [Martin96]

languages

• generic simulation languages
  • introduce entities, messages and simulation time semantics
  • event and state constraints allow optimization
  • both event and process oriented
  • e.g. Simula [Dahl66], Parsec [Bagrodia98] / GloMoSim [Zeng98]

• application-specific languages
  • e.g. Apostle [Bruce97], TeD [Perumalla98]

• simulation libraries
  • move functionality to user-space for performance; monolithic prog.
  • usually event-oriented
  • e.g. Yansl [Joines94], Compose [Martin95], ns2 [McCanne95]
virtual machine-based simulation

- **Proposal:**

  A virtual machine-based simulator benefits from the advantages of both the traditional systems and language-based designs by leveraging standard compilers and language runtimes as well as ensuring efficient simulation execution through transparent cross-cutting program transformations and optimizations.

- **JiST – Java in Simulation Time**
  - converts a virtual machine into a simulation platform
  - no new language, no new library, no new runtime
  - merges modern language and simulation semantics
    - combines systems-based and languages-based approaches
    - result: virtual machine-based simulation

<table>
<thead>
<tr>
<th></th>
<th>kernel</th>
<th>library</th>
<th>language</th>
<th>JiST</th>
</tr>
</thead>
<tbody>
<tr>
<td>transparent</td>
<td>++</td>
<td></td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>efficient</td>
<td></td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>standard</td>
<td>++</td>
<td>++</td>
<td></td>
<td>++</td>
</tr>
</tbody>
</table>
system architecture

1. Compile simulation with standard Java compiler
2. Run simulation within JiST (within Java); simulation classes are dynamically rewritten to introduce simulation time semantics:
   • extend the Java object model and execution model
   • instructions take zero (simulation) time
   • time explicitly advanced by the program: \texttt{sleep(time)}
   • progress of time is dependent on program progress
3. Rewritten program interacts with simulation kernel
jist object model

- program state contained in **objects**
- objects contained in **entities**
  - think of an entity as a simulation component
  - an entity is any class tagged with the `Entity` interface
  - each entity runs at its own simulation **time**
  - as with objects, entities do not share state
  - akin to JKernel [Hawblitzel98] process in spirit, without the threads!
jist execution model

- Entity methods are an event interface
  - Simulation time invocation
  - Non-blocking; invoked at caller entity time; no continuation
  - Like co-routines, but scheduled in simulation time
- Entity references replaced with separators
  - Event channels; act as state-time boundary
  - Demarcate a TimeWarp-like process, but at finer granularity
a basic example

- the “hello world” of event simulations

```java
class HelloWorld implements JistAPI.Entity {
    public void hello()
    {
        JistAPI.sleep(1);
        hello();
        System.out.println("hello world, " +
                      "time=" + JistAPI.getTime());
    }
}
```

- demo!

<table>
<thead>
<tr>
<th>Java</th>
<th>JiST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack overflow @hello</td>
<td>hello world, time=1</td>
</tr>
<tr>
<td></td>
<td>hello world, time=2</td>
</tr>
<tr>
<td></td>
<td>hello world, time=3 etc.</td>
</tr>
</tbody>
</table>
jist micro-benchmark: event throughput

<table>
<thead>
<tr>
<th>Simulation event throughput</th>
<th>Simulation event throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>time (seconds)</td>
<td>time (seconds)</td>
</tr>
<tr>
<td># of events (in millions)</td>
<td># of events (in millions)</td>
</tr>
</tbody>
</table>

5x10^6 events

<table>
<thead>
<tr>
<th></th>
<th>time (sec)</th>
<th>vs. reference</th>
<th>vs. JIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference</td>
<td>0.74</td>
<td></td>
<td>0.76x</td>
</tr>
<tr>
<td>JIST</td>
<td>0.97</td>
<td>1.31x</td>
<td>1.97x</td>
</tr>
<tr>
<td>Parsec</td>
<td>1.91</td>
<td>2.59x</td>
<td>3.36x</td>
</tr>
<tr>
<td>ns2-C</td>
<td>3.26</td>
<td>4.42x</td>
<td>9.84x</td>
</tr>
<tr>
<td>GloMoSim</td>
<td>9.54</td>
<td>12.93x</td>
<td>78.97x</td>
</tr>
<tr>
<td>ns2-Tcl</td>
<td>76.56</td>
<td>103.81x</td>
<td></td>
</tr>
</tbody>
</table>
**jist micro-benchmark: memory overhead**

### Memory Overhead Per Entity and Event

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Memory per Entity (B)</th>
<th>Memory per Event (B)</th>
<th>Memory for 10K Nodes (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JiST</td>
<td>36</td>
<td>36</td>
<td>21</td>
</tr>
<tr>
<td>GloMoSim</td>
<td>36</td>
<td>64</td>
<td>35</td>
</tr>
<tr>
<td>ns2 *</td>
<td>544</td>
<td>40</td>
<td>74</td>
</tr>
<tr>
<td>Parsec</td>
<td>28536</td>
<td>64</td>
<td>2885</td>
</tr>
</tbody>
</table>

### Simulation Entity Memory Overhead

![Graph showing memory overhead per entity](image1)

### Simulation Event Memory Overhead

![Graph showing memory overhead per event](image2)
**SWANS**

- **Scalable Wireless Ad hoc Network Simulator**
  - similar functionality to ns2 [McCanne95] and GloMoSim [Zeng98], but...
  - runs standard Java network applications over simulated networks
  - can simulate networks of **1,000,000 nodes** sequentially, on a single commodity uni-processor
  - runs on top of JiST; SWANS is a JiST application
  - uses hierarchical binning for efficient propagation
  - component-based architecture written in Java

![Diagram of SWANS simulation stack]

<table>
<thead>
<tr>
<th></th>
<th>files</th>
<th>classes</th>
<th>lines</th>
<th>semi</th>
</tr>
</thead>
<tbody>
<tr>
<td>JIST</td>
<td>29</td>
<td>117</td>
<td>14256</td>
<td>3530</td>
</tr>
<tr>
<td>SWANS</td>
<td>85</td>
<td>220</td>
<td>29157</td>
<td>6586</td>
</tr>
<tr>
<td>Other</td>
<td>32</td>
<td>80</td>
<td>7204</td>
<td>2525</td>
</tr>
<tr>
<td>Total</td>
<td>146</td>
<td>417</td>
<td>50617</td>
<td>12641</td>
</tr>
</tbody>
</table>
## SWANS performance

### Time for 2 minute NDP simulation

<table>
<thead>
<tr>
<th>Network Size (Nodes)</th>
<th>SWANS (hier)</th>
<th>SWANS (scan)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1s</td>
<td>10s</td>
</tr>
<tr>
<td>10^2</td>
<td>10s</td>
<td>1m</td>
</tr>
<tr>
<td>10^3</td>
<td>1m</td>
<td>10m</td>
</tr>
<tr>
<td>10^4</td>
<td>10m</td>
<td>1h</td>
</tr>
<tr>
<td>10^5</td>
<td>1h</td>
<td>1h</td>
</tr>
<tr>
<td>10^6</td>
<td>1h</td>
<td>1h</td>
</tr>
</tbody>
</table>

### SWANS-hier NDP simulation

<table>
<thead>
<tr>
<th>t=2m</th>
<th>nodes</th>
<th>initial memory</th>
<th>avg. memory</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10,000</td>
<td>13 MB</td>
<td>45 MB</td>
<td>2 m</td>
</tr>
<tr>
<td></td>
<td>100,000</td>
<td>100 MB</td>
<td>160 MB</td>
<td>25 m</td>
</tr>
<tr>
<td></td>
<td>1 million</td>
<td>1000 MB</td>
<td>1200 MB</td>
<td>5.5 h</td>
</tr>
<tr>
<td></td>
<td>20,000,000</td>
<td>1000 MB</td>
<td>1200 MB</td>
<td>20 ms</td>
</tr>
</tbody>
</table>

- **Virtual machine-based simulation**
benefits of the jist approach

more than just performance...

- **application-orientated benefits**
  - type safety: source and target statically checked
  - event types: not required (implicit)
  - event structures: not required (implicit)
  - debugging: dispatch source location and state available

- **language-orientated benefits**
  - Java: standard language, compiler, runtime
  - garbage collection: cleaner code, memory savings
  - reflection: script-based simulation configuration
  - safety: fine grained isolation
  - robustness: no memory leaks, no crashes

- **system-orientated benefits**
  - IPC: no context switch, no serialization, zero-copy
  - Java kernel: cross-layer optimization
  - rewriting: no source-code access required, cross-cutting program transformations and optimizations
  - distribution: provides a single system image abstraction
  - concurrency: model supports parallel and speculative execution

- **hardware-orientated benefits**
  - cost: COTS hardware and clusters
  - portability: runs on everything
**rewriter flexibility**

- **simulation time transformation**
  - extend Java object model with entities
  - extend Java execution model with events
  - language-based simulation kernel

- **extensions to the model**
  - **timeless objects**: pass-by-reference to avoid copy, saves memory
  - **reflection**: scripting, simulation configuration, tracing
  - **tight event coupling**: cross-layer optimization, debugging
  - **proxy entities**: interface-based entity definition
  - **blocking events**: call and callback, CPS transformation, standard applications
  - **simulation time concurrency**: Threads, Channels and other synch. primitives
  - **distribution**: location independence of entities, single system image abstraction
  - **parallelism**: concurrent and speculative execution
  - **orthogonal additions, transformations and optimizations**

- **platform for simulation research**
  - e.g. reverse computations in optimistic simulation [Carothers99]
  - e.g. stack-less process oriented simulation [Booth97]
summary

• JiST – Java in Simulation Time

• prototype virtual machine-based simulation platform
• merges systems and language-based approaches

<table>
<thead>
<tr>
<th></th>
<th>kernel</th>
<th>library</th>
<th>language</th>
<th>JiST</th>
</tr>
</thead>
<tbody>
<tr>
<td>transparent</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td></td>
</tr>
<tr>
<td>efficient</td>
<td></td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>standard</td>
<td>++</td>
<td>++</td>
<td></td>
<td>++</td>
</tr>
</tbody>
</table>

• runs SWANS: Scalable Wireless Ad hoc Network Simulator

• efficient: both in terms of throughput and memory
• flexible: timeless objects, reflection-based scripting, tight event coupling, proxy entities, continuations and blocking methods, simulation time concurrency, distribution, concurrency ... serve as a research platform
JiST – Java in Simulation Time

An efficient, unifying approach to simulation using virtual machines

THANK YOU.

http://jist.ece.cornell.edu/