SimuBoost: Scalable Parallelization of Functional System Simulation
Marc Rittinghaus, Konrad Miller, Marius Hillenbrand, Frank Bellosa

Motivation
- Want: Operating system performance analysis
- Need: Functional full system simulation to monitor system non-intrusively
- Challenges: Functional system simulation too slow for long-running workloads.

<table>
<thead>
<tr>
<th>Virtualization</th>
<th>Simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>KVM</td>
<td>QEMU</td>
</tr>
<tr>
<td>~ 1x</td>
<td>~ 100x</td>
</tr>
<tr>
<td>Average slowdown for: Kernel build, SPECint_base2006, LAMMPS</td>
<td></td>
</tr>
</tbody>
</table>

Loss of interaction with non-simulated remote hosts.

Existing Techniques
- Sample and extrapolate
  (Sherwood et al. SimPoints)
  Not all applications show phase behavior (gcc) (Weaver et al.).
  Less probable for whole system.
  How to find phases without using simulation first?
- Parallel multi-core simulation
  (Sing et al. POFMU)
  Only scales in number of simulated CPUs.
- Reduce workload
  (KinnOwski et al. MinibSPEC)
  Not always possible.

Goal: Scale-out single-core functional full system simulation

Basic Idea
- (1) Split simulation time
- (2) Simulate intervals in parallel
  - Scales with the run-time of the workload.
  - Applicable to single-core simulations.

How to bootstrap the simulation of [2...n]?

Approach
- Run workload in virtual machine
  - Preserves interactivity and network connectivity.
- Create checkpoints at interval boundaries to bootstrap simulations
- Run simulations in parallel
  - Distribute jobs across machines.

Speedup and Scalability
- Speedup depends on speed difference between virtualization and simulation, and interval length
- Minimize virtualization overhead...
  (logging and checkpointing)
  ...and calculate optimal interval length from speed difference and overheads.
- Predicted speedup for 1h workload: 84x @ 90 nodes (94% parallel efficiency)
  100x slowdown, 100ms downtime/checkpoint [Sun et al. Remus], 8% logging overhead, 1s start-up delay

Lightweight Checkpointing
- Goals:
  - Short downtime, small checkpoints
  - Easy & fast access
- Copy-on-write checkpointing
  - Resume VM before saving memory & HDD.
- Incremental, hash-based checkpointing
  - Deduplicate within and across checkpoints.
  - Of modified data, we can deduplicate:
    - RAM pages: 5%-40%
    - Disk blocks: 35%-80%
- Access via key-value store
  - Store <hash, page/block data>-pairs.
  - Checkpoint = list of hashes.

Functional Continuity
- Virtualization introduces non-determinism
  - Different I/O timing and data between stages.
  - Virtualization and simulation drift apart
  - Virtualization and simulation stay synchronized
  - (1) Trap and log non-deterministic events
  - (2) Precisely replay events in the simulation
  - (Ding et al. PQEMU)
  - Overhead: <8%

Preserving Functional Continuity
- Virtualization introduces non-determinism
  - Different I/O timing and data between stages.
  - Virtualization and simulation drift apart
  - Virtualization and simulation stay synchronized

Functional Continuity
- Virtualization introduces non-determinism
  - Different I/O timing and data between stages.
  - Virtualization and simulation drift apart
  - Virtualization and simulation stay synchronized
  - (1) Trap and log non-deterministic events
  - (2) Precisely replay events in the simulation
  - (Ding et al. PQEMU)
  - Overhead: <8%

Preserving Functional Continuity
- Virtualization introduces non-determinism
  - Different I/O timing and data between stages.
  - Virtualization and simulation drift apart
  - Virtualization and simulation stay synchronized
  - (1) Trap and log non-deterministic events
  - (2) Precisely replay events in the simulation
  - (Ding et al. PQEMU)
  - Overhead: <8%

Preserving Functional Continuity
- Virtualization introduces non-determinism
  - Different I/O timing and data between stages.
  - Virtualization and simulation drift apart
  - Virtualization and simulation stay synchronized
  - (1) Trap and log non-deterministic events
  - (2) Precisely replay events in the simulation
  - (Ding et al. PQEMU)
  - Overhead: <8%

Preserving Functional Continuity
- Virtualization introduces non-determinism
  - Different I/O timing and data between stages.
  - Virtualization and simulation drift apart
  - Virtualization and simulation stay synchronized
  - (1) Trap and log non-deterministic events
  - (2) Precisely replay events in the simulation
  - (Ding et al. PQEMU)
  - Overhead: <8%