Towards Scalable Parallelization of Functional System Simulation with SimuBoost

GI Fachgruppentreffen Betriebssysteme (BS) 2016
Marc Rittinghaus, Frank Bellosa
Motivation

- Study properties of redundant memory contents [Miller13]
  - Origin? Lifetime? Sharing possible?
  - Analyze memory contents after each modification
  - But: Analysis should not affect workload

- Analyze memory access patterns on system interfaces [Jurczyk13, Wilhelm15]
  - Detect vulnerabilities in Windows 8 and Xen (CVE-2015-8550)
  - Trace individual memory reads and writes

We want detailed runtime information
Motivation

- Operating system research
  - Debugging
  - Application, OS, and hardware interaction
  - Malware and vulnerabilities

- Functional Full System Simulation
  - But: It is slow

<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>
</tbody>
</table>

Average slowdowns for: kernel build, SPECint_base06, LAMMPS

- Not practical for long-running workloads
- Loss of interactivity (users and remote hosts)
Basic Acceleration Approach

(1) Split simulation into time intervals
(2) Simulate intervals simultaneously
- Does not trade accuracy for speed
- Applicable to single-CPU simulations
- Scales with run-time of workload

• How to bootstrap the simulation of $i[1..n]$?
  • Still no interactivity
SimuBoost

- Leverage fast virtualization
  - Checkpoints at interval boundaries bootstrap simulations
  - Hardware acceleration provides full interactivity
  - Speed difference drives parallelization
Leverage fast virtualization
- Checkpoints at interval boundaries bootstrap simulations
- Hardware virtualization provides full interactivity
- Speed difference drives parallelization

Challenges: Preserve interactivity and speedup
Stop-And-Copy

$s_i[k] \rightarrow s_i[k+1]$
Stop-And-Copy

\[ i[k] \rightarrow i[k+1] \]

Virtualization

VM RAM

Checkpoint
Stop-And-Copy

\[ i[k] \rightarrow i[k+1] \]

\textit{suspended} Virtualization

Checkpoint
Stop-And-Copy

- Downtime depends on VM size
- Not suited for interactive use
- Limited parallelization

We need to drastically speedup checkpointing
Incremental Stop-And-Copy

Observation: Only some data modified per interval

<table>
<thead>
<tr>
<th>pts_build_linux_kernel</th>
<th>spec_jbb</th>
</tr>
</thead>
<tbody>
<tr>
<td>22000 pages/s (85 MiB/s)</td>
<td>53000 pages/s (200 MiB/s)</td>
</tr>
</tbody>
</table>
Incremental Stop-And-Copy

- Idea: Save only modified data
  - Track dirty pages via page protections
  - Use previous checkpoints to get unmodified data
Incremental Stop-And-Copy

- Idea: Save only modified data
  - Track dirty pages via page protections
  - Use previous checkpoints to get unmodified data
Incremental Stop-And-Copy

- Reduced downtime
  - Less dependent on VM size

![Diagram showing downtime reduction and saved downtime intervals.](image)

![Graph showing reduced downtime vs. memory size.](image)
Incremental Stop-And-Copy

- Reduced downtime
  - Less dependent on VM size
- But: Downtime depends on
  - Interval length
  - Workload
**Incremental Stop-And-Copy**

- Reduced downtime
  - Less dependent on VM size
- But: Downtime depends on
  - Interval length
  - Workload
- But: Downtime strongly fluctuates

We need to further speedup checkpointing
Incremental Copy-On-Write

Virtualization

$\delta[k] \rightarrow \delta[k+1]$
Incremental Copy-On-Write

Idea: Save modified pages asynchronously
- Use write-protection to prevent modification
Incremental Copy-On-Write

- Idea: Save modified pages asynchronously
  - Use write-protection to prevent modification
Incremental Copy-On-Write

- Idea: Save modified pages asynchronously
- Use write-protection to prevent modification
- Copy and release protection on pagefault
Incremental Copy-On-Write

- Drastically reduced downtime
  - Pagefaults do not impede interactivity
- Less dependent on
  - Interval length
  - Workload
Incremental Copy-On-Write

- Drastically reduced downtime
  - Pagefaults do not impede interactivity
- Less dependent on
  - Interval length
  - Workload
- Almost constant downtime

We can do checkpointing fast enough
Checkpoint Distribution – The Naïve Way

- Nodes request full checkpoints from central server

But: Central server becomes bottleneck
- Limits parallelization and speedup
SimuBoost Evaluation

- Prototype: 1GiB RAM, 1s intervals, 4 simulation nodes
  - SimuBoost delivers predicted speedup [Rittinghaus13]
  - But: Saturates 10 Gbit Ethernet

Need to avoid single bottleneck
Future Checkpoint Distribution

- Idea: Only send new data
- Deduplicate and compress data

<table>
<thead>
<tr>
<th>pts_build_linux_kernel</th>
<th>spec_jbb</th>
</tr>
</thead>
<tbody>
<tr>
<td>22000 pages/s (85 MiB/s)</td>
<td>53000 pages/s (200 MiB/s)</td>
</tr>
<tr>
<td>5000 pages/s (20 MiB/s)</td>
<td>16000 pages/s (65 MiB/s)</td>
</tr>
</tbody>
</table>

- Use distributed file system (e.g., Ceph [Weil06])
- Append new data to global file
- Checkpoint = Map of VM addresses to offsets in file
Conclusion

- Slowdown of Functional Full System Simulation: >100x

- SimuBoost: Accelerate simulation
  - Run workload with fast virtualization
  - Take checkpoints in regular intervals
  - Start parallel simulations on checkpoints

Challenges

- Fast checkpoint creation ✔️
  - Incremental Copy-On-Write

- Fast checkpoint distribution ✔️
  - Distributed file system
Deterministic Replay

- (1) Trap and log non-deterministic events in the hypervisor
- (2) Precisely replay events in the simulation

Non-deterministic events (e.g., interrupts, timing instructions)
- …appear at equal points in the instruction stream
- …produce same data output

Existing work: Retrace [Sheldon07], V2E [Yan12]
### Speedup and Scalability

- **Right interval length is crucial**
  - Too short (a):
    - Checkpoint time dominates
  - Too long (c):
    - Little parallelization
    - Long simulation of final interval

- **Example scenario:**
  - 100ms downtime, 8% logging, 100x slowdown
  - Optimal interval length: 2s
  - Best possible speedup for 1h workload:
    84x @ 90 nodes (94% parallel efficiency)

#### Near linear speedup possible
Selected Previous Research

- Workload Reduction
  - MinneSPEC [KleinOsowski02]

- Simulate samples and extrapolate
  - Truncated Execution
  - SimPoints [Sherwood02]
  - SMARTS [Wunderlich03]

- Improve simulation engine
  - Optimize engine: below 5x speedup mark
  - Parallelize simulation of vCPUs [Ding11]

- Divide simulation time
  - For microarchitectural simulations: DiST [Girbal03]
References