Exploring VM Introspection:
Techniques and Trade-offs

VEE
March 15, 2015

Sahil Suneja
Eyal de Lara
University of Toronto

Canturk Isci
Vasanth Bala
IBM T.J. Watson Research
Virtual Machine Introspection: Uses
VMI Techniques - Expose VM State

**XEN**
- XenGuestAgent
dump-core
xc_map_foreign_range()
gdbsx
...

**QEMU/KVM**
- pmemsave
  migrate
  hugetlbfs
  /proc/pid/mem
  ...

**VMWare**
- VMSafe()
vms2core
  .vmem
  ...

**LIBRARY / 3rd PARTY**
- Libvirt
  LibVMI
  ...

*Technique Vs. Technique?*
- Latency, overhead, complexity, ...
# Qualitative Comparison

<table>
<thead>
<tr>
<th>Properties</th>
<th>Live</th>
<th>View Consistency</th>
<th>Speed</th>
<th>Resource Cost</th>
<th>VM perf impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest cooperation / agent assisted access</td>
<td>✔</td>
<td>✔</td>
<td>⬤</td>
<td>⬤</td>
<td>🟥</td>
</tr>
<tr>
<td>Halt Snap</td>
<td>✔</td>
<td>✔</td>
<td>⬤</td>
<td>⬤</td>
<td>🟥</td>
</tr>
<tr>
<td>Live Snap</td>
<td>✔</td>
<td>✔</td>
<td>⬤</td>
<td>⬤</td>
<td>🟥</td>
</tr>
<tr>
<td>Live Memory Mapping</td>
<td>✔</td>
<td>✔</td>
<td>⬤</td>
<td>⬤</td>
<td>🟥</td>
</tr>
<tr>
<td>Live Memory Reads</td>
<td>✔</td>
<td>✔</td>
<td>⬤</td>
<td>⬤</td>
<td>🟥</td>
</tr>
<tr>
<td>Guest-Halting Memory Map and Reads</td>
<td>✔</td>
<td>✔</td>
<td>⬤</td>
<td>⬤</td>
<td>🟥</td>
</tr>
<tr>
<td>Live Interface Access</td>
<td>✔</td>
<td>✔</td>
<td>⬤</td>
<td>⬤</td>
<td>🟥</td>
</tr>
<tr>
<td>Guest Halting Interface Access</td>
<td>✔</td>
<td>✔</td>
<td>⬤</td>
<td>⬤</td>
<td>🟥</td>
</tr>
</tbody>
</table>
Characterizing VMI Techniques: Dimensions

- Snapshotting
- Guest Cooperation
- Memory Access Type
- Guest Liveness
Proposing VMI Taxonomy

- **Guest Cooperation**
  - Yes
  - Agent assisted access

- **Snapshotting**
  - Yes
  - Live
  - Live Snap
  - Memory Access Type
    - Map
    - Live memory map
  - Interface based
    - Live interface access
  - Read
    - Live reads
  - No
  - Halt Snap

- **Memory Access Type**
  - Interface based
  - Map / Read
  - Guest-halting interface access
  - Guest-halting map / read
Quantitative Comparison: Use-case

- Periodic generic monitoring via VMI
  - In-memory kernel data structure traversal
    - `task_struct`, `mm_struct`, `files_struct`, `net_devices` ...
  - Extract 700KB volatile VM memory state

<table>
<thead>
<tr>
<th>Category</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
<td>NumCores, Hz, CacheSize, ...</td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td>Nodename, Release, Arch, ...</td>
</tr>
<tr>
<td><strong>N/W device</strong></td>
<td>HWaddr, Ipaddr, TX/RX bytes, ...</td>
</tr>
<tr>
<td><strong>Modules</strong></td>
<td>Name, State, ...</td>
</tr>
<tr>
<td><strong>Process</strong></td>
<td>PID, Command, RSS, ...</td>
</tr>
<tr>
<td><strong>Open files</strong></td>
<td>FD → filename, ...</td>
</tr>
<tr>
<td><strong>Memory Mapping</strong></td>
<td>MappedFiles, VA → PA mappings, ...</td>
</tr>
<tr>
<td><strong>N/W connections</strong></td>
<td>SocketState, {Src, Dst, Ports}, ...</td>
</tr>
</tbody>
</table>
Quantitative Comparison: Dimensions

- Maximum frequency of monitoring?
- Resource usage cost on host?
- Impact on VM’s workload?
Maximum Monitoring Frequency

- Live interface access
- Guest-halting interface access
- Halt Snap | Memory dump
- Halt Snap | In-mem snapshot
- Guest-halting reads | Semilive
- Live reads | Memory xfer channel
- Live reads | Direct memory access
- Live map | Host phys mem access
- Live map | File-backed VM mem

Monitoring Frequency [Hz]
1. Better performance does not come at an added cost
2. Normalized CPU cost per Hz
3. Lower CPU usage for halting-reads misleading
Impact on VM Perf: x264 CPU Benchmark [1/2]

+ Individual techniques' performance
+ VM impact vs. Technique resource allotment
+ Memory, N/W, Disk benchmarks
Impact on VM’s Perf: httpperf Webserver [2/2]

1. Lower maximum monitoring frequencies: \( \text{apache} \Rightarrow \text{in-VM state} \)
2. Backlog of pending requests in wait queue
3. >100% degradation on response times with halt snap

\[ + \text{ in paper} \]
Part II: Consistency of VM State
Consistency of VM State [1/3]

- Missing OS-context within VMI scope
  - Inconsistency in observed data structures

- Common solution: pause-and-introspect (PAI)

- Goal:
  - What are these inconsistencies?
  - How often do these occur?
  - Is PAI helpful?
Consistency of VM State [2/3]

- Observation #1: Multiple forms of inconsistencies

**Intrinsic Inconsistencies**
- Zombie tasks
- Dying tasks
- As-good-as-dead tasks
- Fresh tasks

**Extrinsic Inconsistencies**
- Task dies during introspection
- Attributes change during introspection
Observation #2: Inconsistencies are rare

- Intrinsic
  - Zombie tasks
  - Dying tasks
  - As-good-as-dead
- Extrinsic
  - Fresh tasks
  - Task dies during introspection
  - Attributes change during introspection

Fork @ 10Hz, process lifespan 1s

Observation #3: Expensive PAI does not mitigate all inconsistencies
- Intrinsic remain
Conclusion

- **Taxonomy** to organize existing VMI techniques
  - Guest Cooperation, Snapshotting, Guest Liveness, Memory Access Type

- **Comparative evaluation** of techniques
  - Quantitative: Operating frequencies, Resource consumption on host, and Overheads on target systems.
  - + Qualitative: Liveness, Consistency, Compatibility, ...

- **Memory inconsistency analysis**
  - Multiple forms of inconsistencies
  - Inconsistencies are rare
  - PAI have marginal benefits, despite high cost
BACKUP SLIDES -->
## Qualitative Comparison

<table>
<thead>
<tr>
<th>Properties</th>
<th>Live</th>
<th>View Consistency</th>
<th>Speed</th>
<th>Resource Cost</th>
<th>VM perf impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest cooperation / agent assisted access</td>
<td>✓</td>
<td>✓</td>
<td>⚫⚫⚫</td>
<td>⚫⚫⚫</td>
<td>⚫⚫⚫</td>
</tr>
<tr>
<td>Halt Snap</td>
<td>✓</td>
<td>✓</td>
<td>⚫⚫⚫</td>
<td>⚫⚫⚫</td>
<td>⚫⚫⚫</td>
</tr>
<tr>
<td>Live Snap</td>
<td>✓</td>
<td>✓</td>
<td>⚫⚫⚫</td>
<td>⚫⚫⚫</td>
<td>⚫⚫⚫</td>
</tr>
<tr>
<td>Live Memory Mapping</td>
<td>✓</td>
<td>✓</td>
<td>⚫⚫⚫⚫</td>
<td>⚫</td>
<td>⚫⚫⚫</td>
</tr>
<tr>
<td>Live Memory Reads</td>
<td>✓</td>
<td>✓</td>
<td>⚫⚫⚫</td>
<td>⚫⚫⚫</td>
<td>⚫⚫⚫</td>
</tr>
<tr>
<td>Guest-Halting Memory Map and Reads</td>
<td>✓</td>
<td>✓</td>
<td>⚫⚫⚫</td>
<td>⚫⚫⚫</td>
<td>⚫⚫⚫</td>
</tr>
<tr>
<td>Live Interface Access</td>
<td>✓</td>
<td>✓</td>
<td>⚫</td>
<td>⚫⚫⚫⚫</td>
<td>⚫⚫⚫</td>
</tr>
<tr>
<td>Guest Halting Interface Access</td>
<td>✓</td>
<td>✓</td>
<td>⚫</td>
<td>⚫⚫⚫⚫</td>
<td>⚫⚫⚫</td>
</tr>
</tbody>
</table>
Observations and Recommendations

- Broad Spectrum of Choices
- Guest Cooperation vs. Out-of-band
- VMI use-case
- VM Workload
- Host / Hypervisor Specialization
- Mapping over direct reads
- Guest-halting map/reads over halting snapshots
- Consistency vs. Liveness, Realtimeness, and VM perf.
- Monitoring Overhead vs. Resource Usage
- Scalability of approaches