Out of Hypervisor (OoH): Efficient Dirty Page Tracking In Userspace Using Hardware Virtualization Features

Stella Bitchebe
(bitchebe@i3s.unice.fr)

Alain Tchana
(alain.tchana@grenoble-inp.fr)

November 17th, 2022

UNIVERSITÉ CÔTE D'AZUR

ENS DE LYON

Virtualized Clouds: Dirty Page Tracking in Userspace

Purpose

- WSS (working set size) estimation (*for memory overcommitment*)
- Live migration (*for maintenance*)
- Checkpointing (*for recovery after failure*)
- Garbage collection (*for better memory management*)

Virtualized Clouds: Dirty Page Tracking in Userspace

Purpose

- WSS (working set size) estimation (*for memory overcommitment*)
- Live migration (*for maintenance*)
- Checkpointing (*for recovery after failure*)
- Garbage collection (*for better memory management*)

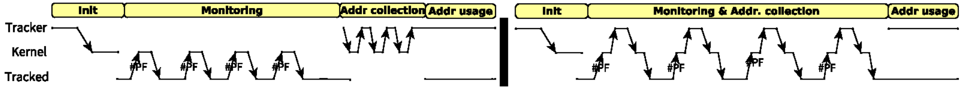
Nomenclature

- Tracker: the monitoring thread (e.g., CRIU, Boehm GC)
- Tracked: the thread whose memory is monitored (any application)

Virtualized Clouds: Dirty Page Tracking in Userspace

Current approach

- Page write protection
- Two main solutions
  - Linux /proc interface
  - Linux userfaultfd (ufd) interface



(a) /proc based tracking

(b) userfaultfd based tracking

1. Virtualized Clouds: Dirty Page Tracking in Guest Userspace
2. **Problem: Limits of Existing Solutions**
3. Solution: Hardware-Assisted Virtualization Out of Hypervisor (OoH)
4. OoH for PML
5. Evaluatons
6. Conclusion

Problem: Limits of Page Write Protextion

Overhead

- ufd: Page fault (#PF) handling and context switches
  - **15.6**× and **14.5**× slowdown for 1GB on Tracked and Tracker respectively

Problem: Limits of Page Write Protextion

Overhead

- ufd: Page fault (#PF) handling and context switches
  - **15.6**× and **14.5**× slowdown for 1GB on Tracked and Tracker respectively
- /proc: #PF handling and page table (PT) walks
  - ∼**2.234ms**: parse PT and flush TLB (in the kernel)
  - ∼**594.187ms**: parse PT in userspace (/proc/PID/pagemap) for 1GB
  - **4.3**× and **2.5**× slowdown for 1GB on Tracked and Tracker respectively

Virtualization Technologies

- Goal: reduce overheads of virtualization
- AMD-v (2006) and Intel VT (2005)
  - CPU virtualization (e.g., VT-x)
  - MMU virtualization (e.g., EPT)
  - I/O virtualization (e.g., SRIOV)

Intel VT Features Categorization

2 main groups:

G_1: Multiplexing Features

- Extended Page Table (EPT)
- Single Root I/O Virtualization (SRIOV)
- Advanced Programmable Interrupt Controller virtualization (APICv)

Intel VT Features Categorization

2 main groups:

G_1: Multiplexing Features

- Extended Page Table (EPT)
- Single Root I/O Virtualization (SRIOV)
- Advanced Programmable Interrupt Controller virtualization (APICv)

G_2: Management Features

- Page Modification Logging (PML)
- Sub-Page write Permissions (SPP)
- Cache Allocation Technology (CAT)

Intel VT Features Categorization

2 main groups:

G_1: Multiplexing Features

- Extended Page Table (EPT)
- Single Root I/O Virtualization (SRIOV)
- Advanced Programmable Interrupt Controller virtualization (APICv)

G_2: Management Features

- Page Modification Logging (PML)
- Sub-Page write Permissions (SPP)
- Cache Allocation Technology (CAT)

G_2's features can be exploited in VMs

OoH Principle

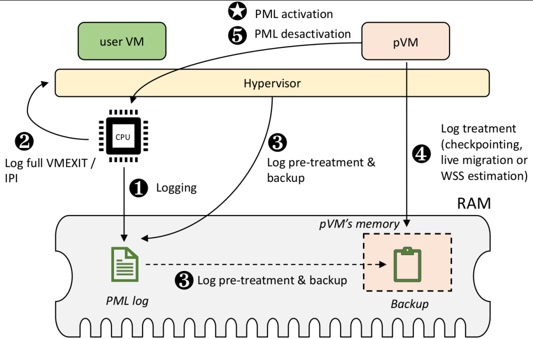
- **New research axis**
- Objective
  - Make some hardware virtualization features usable within the guest OS
  - From conception/design of features

OoH Principle

- **New research axis**
- Objective
  - Make some hardware virtualization features usable within the guest OS
  - From conception/design of features
- Methodology
  - Kernel module and userspace library
  - Hypercalls and event channels between hypervisor and guests
  - Leverage existing extensions for direct passthrough
  - Hardware changes (e.g., ISA extension)

PML Functioning

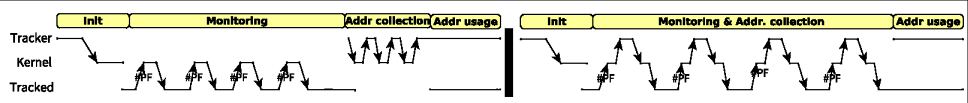
Allows the hypervisor to track guest memory accesses



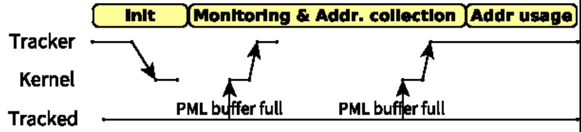
Intel PML in the OoH Context

- To accelerate CRIU checkpointing and Boehm garbage collection

PML-based Dirty Page Tracking in Userspace



(a) /proc based tracking
(b) userfaultfd based tracking



(c) OoH-PML based tracking

OoH for PML

Challenges

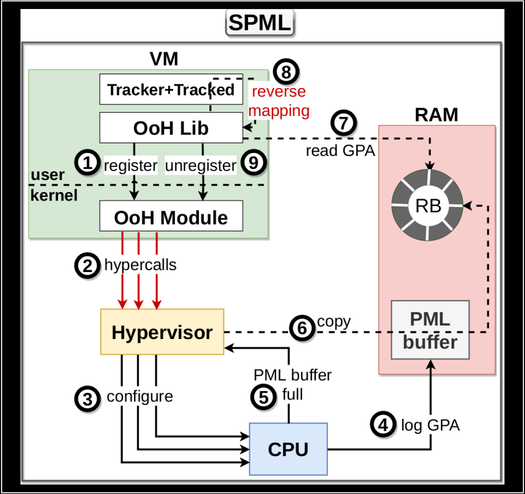
- (C_1) PML can only be managed by the hypervisor
- (C_2) PML works at coarse-grained, that is it concerns the entire VM
- (C_3) PML only logs GPAs

OoH for PML

Two Solutions

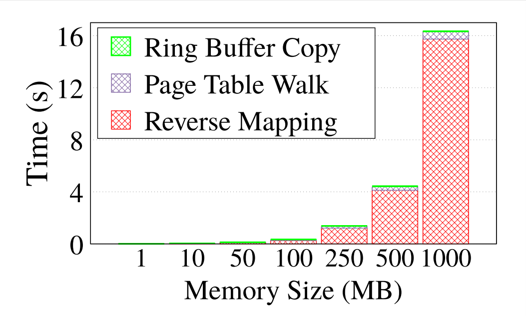
- Shadow PML (SPML): no hardware modification
  - Its significant overhead justifies EPML
- Extended PML (EPML): modest hardware changes

Shadow PML (SPML): Design



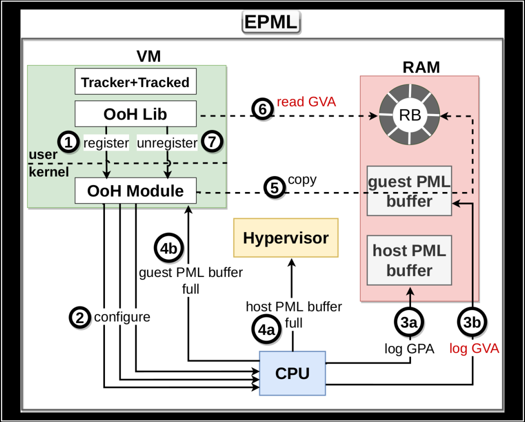
Shadow PML (SPML): Limitations

- Costly reverse mapping (∼15.739 s for 1GB working set)

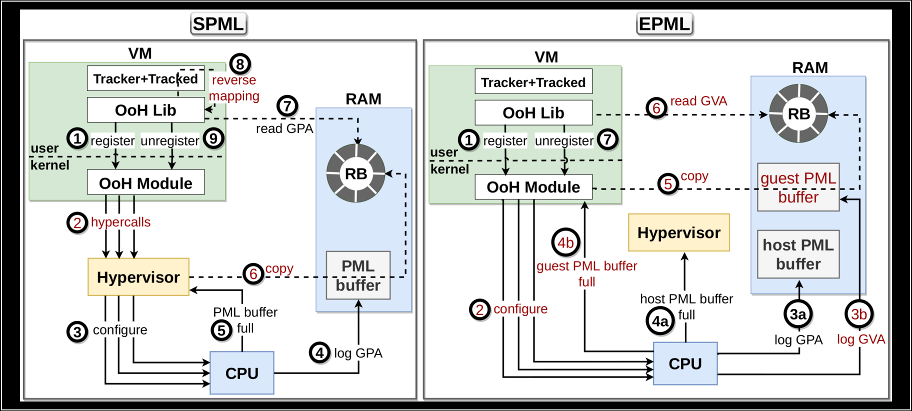


- Costly hypercalls (4.49μs for empty hypercall)

Extended PML (EPML): Design



Extended PML (EPML): Design



OoH Security and Isolation

Vis-à-vis the Hypervisor

- Small TCB[1] (194LOC) - at least safe as existing hypercalls
- Guest does not see nor manipulate host physical memory
- Ring buffer allocated from VM's memory

[1] Trust Code Base

OoH Security and Isolation

Vis-à-vis the Hypervisor

- Small TCB[1] (194LOC) - at least safe as existing hypercalls
- Guest does not see nor manipulate host physical memory
- Ring buffer allocated from VM's memory

Between VMs

- Same isolation level
- Ring buffer allocated per VM's address space => no possible inference
- Per process ring buffer and restriction to tracker process only

[1] Trust Code Base

Evaluations: Implementation

- We implemented EPML's hardware changes in BOCHS
- We used Xen as the hypervisor and Linux as the guest OS
- We integrated OoH Lib with:
  - CRIU: Checkpoint/Restore in User space
    - Integrated in OpenVZ, Docker, etc.
    - Based on `/proc` technique
  - Boehm GC: popular C/C++ garbage collector
    - Included in Mozilla, GNU Java Compiler, etc.
    - Based on `/proc` technique

Evaluations: Benchmarks

- Macro-benchmarks: tkrzw applications (key value store) and Phoenix applications (MapReduce)
- Three working set sizes (Small, Medium, and Large)

Evaluations: Methodology for EPML

Approach

- Build a formula f
- Show the accuracy of f on other techniques that are measurable

Evaluations: Methodology for EPML

Impact on Tracker

Execution time of Tracker when implementing technique x:
$E(C_{tker}) = E(C_x) + E(C_p) + I(C_x, C_p)$
x: */proc*, *SPML*, *EPML* - C_x: *enable_PML, ring buffer copy, etc.*

Impact on Tracked

Time of Tracked when monitored by a Tracker using technique x:
$E(C_{tked\_tker}) = E(C_{tked}) + E(C_{tker}) + I(C_x, C_{tked})$
$I(C_x, C_{tked})$: *page faults, vmexits, etc.*

Evaluations: Formulas Validation

Metric	Time (ms)
$E(C_{tker})$ **measured**	**5503.79**
$E(C_{tked\_tker})$ **measured**	**135255.35**
$E(C_p)$	251.35
$E(C_{copy\_rb})$	0.49
$E(C_{disable\ pml})$	2.06
$E(C_{rev.\ mapping})$	5419
$E(C_{tker})$ **estimated**	**5672.9**
$E(C_{vmexits})$	18000
N	39
$E(C_{vmread,vmwrite})$	1.73×10^{-3}
$E(C_{tked\_tker})$ **estimated**	**136919.85**

(a) SPML

Metric	Time (ms)
$E(C_{tker})$ **measured**	**1097.99**
$E(C_{tked\_tker})$ **measured**	**115283.98**
$E(C_p)$	251.35
$E(C_{clear\_refs})$	1.409
$E(C_{PTwalk})$	0.89
$E(C_{tker})$ **estimated**	**1116.09**
$E(C_{PFHuser})$	0.27
$E(C_{tked\_tker})$ **estimated**	**114418.58**

(b) /proc

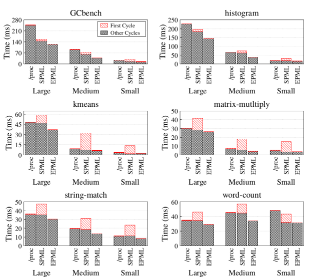
Evaluations: Formulas Validation

SPML accuracy: **96.34%**

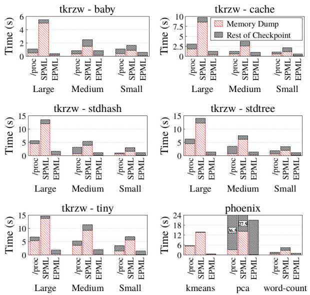
`/proc` accuracy: **99%**

Evaluations: Tracker Results

Boehm



CRIU

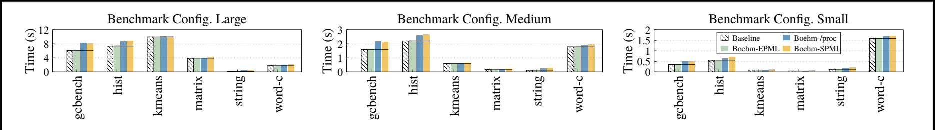


SPML vs. /proc: **5×** and **3×** slowdown resp. on CRIU and Boehm

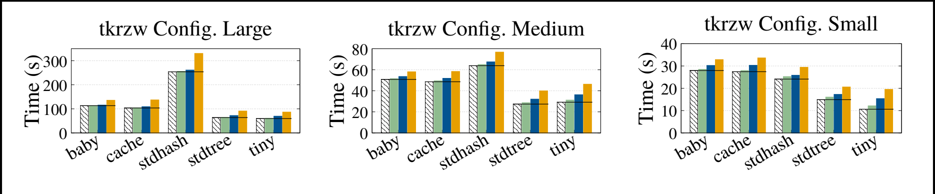
EPML: (**On CRIU:**) **4×** and **13×** speedup compared to /proc and SPML resp. (**On Boehm:**) **2×** and **6×** speedup compared to /proc and SPML resp.

Evaluations: Tracked Results

Boehm



CRIU



Evaluations: Tracked Results

Impact on Tracked

- `/proc`:
  - Up to **102%** overhead on Phoenix-pca with CRIU
  - Up to **232%** overhead on Phoenix `string-match` with Boehm
- **SPML**:
  - Up to **114%** overhead on Phoenix-pca with CRIU
  - Up to **273%** overhead on Phoenix `string-match` with Boehm
- **EPML**:
  - Only **7%** with CRIU
  - Only **24%** with Boehm
  - ==> **16×** improvement

1. Virtualized Clouds: Dirty Page Tracking in Guest Userspace
2. Problem: Limits of Existing Solutions
3. Solution: Hardware-Assisted Virtualization Out of Hypervisor (OoH)
4. OoH for PML
5. Evaluatons
6. **Conclusion**

Conclusion

Dirty Page Tracking

- For wss estimation, live migration, checkpointing, GC, ...
- Induce high overhead on applications

Conclusion

Dirty Page Tracking

- For wss estimation, live migration, checkpointing, GC, ...
- Induce high overhead on applications

OoH for Intel PML (https://github.com/bstellaceleste/OoH)

- For improving process/container checkpointing, concurrent GCs
- **4**$\times$ speedup on Tracker - **16**$\times$ improvement on Tracked

Conclusion

Dirty Page Tracking

- For wss estimation, live migration, checkpointing, GC, ...
- Induce high overhead on applications

OoH for Intel PML (https://github.com/bstellaceleste/OoH)

- For improving process/container checkpointing, concurrent GCs
- **4**× speedup on Tracker - **16**× improvement on Tracked

Take Away

- Existing sofware-based tools can be improved using hardware virtualization features
- Think of OoH from the conception/design of hardware virtualization features

SC22
Dallas, TX | hpc accelerates.

Out of Hypervisor (OoH): Efficient Dirty Page Tracking In Userspace Using Hardware Virtualization Features

Stella Bitchebe
(bitchebe@i3s.unice.fr)

Alain Tchana
(alain.tchana@grenoble-inp.fr)

November 17th, 2022

UNIVERSITÉ CÔTE D'AZUR

ENS DE LYON