Honey, I Shrunk the Guests

Page Access Tracking using a Minimal Virtualisation Layer

30.03.2025

Dustin Tien Nguyen[1], Sebastian Rußer[1], Maximilian Ott[1],
Rüdiger Kapitza[1], Wolfgang Schröder-Preikschat[1], Jörg Nolte[2]

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg
[2] Brandenburgische Technische Universität Cottbus-Senftenberg

FAU Friedrich-Alexander-Universität
Technische Fakultät

btu Brandenburgische
Technische Universität
Cottbus - Senftenberg

Funded by
DFG Deutsche
Forschungsgemeinschaft
German Research Foundation

Project No.
501993201

Plenty of memory, hardly any knowledge

Many memory technologies

- HBM
- DRAM
- Persistent Memory
- CXL-attached memory
- NUMA

Different characteristics

- Capacity
- Latency
- Bandwidth
- Persistence
- Cost

Plenty of memory, hardly any knowledge

Many memory technologies

- HBM
- DRAM
- Persistent Memory
- CXL-attached memory
- NUMA

Different characteristics

- Capacity
- Latency
- Bandwidth
- Persistence
- Cost

How can we efficiently utilise memory?

- ϟ ”Expert interfaces” are difficult (PMDK, libnuma)
- ϟ Language support does not expand to legacy programs (NV-Heaps)

Plenty of memory, hardly any knowledge

Many memory technologies

- HBM
- DRAM
- Persistent Memory
- CXL-attached memory
- NUMA

Different characteristics

- Capacity
- Latency
- Bandwidth
- Persistence
- Cost

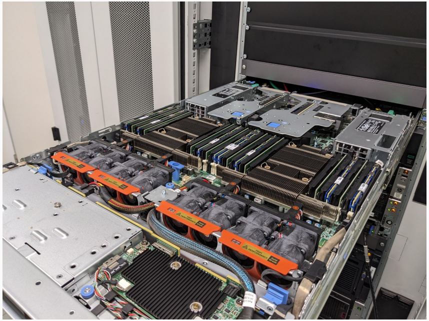
How can we efficiently utilise memory?

- ⚡ ”Expert interfaces” are difficult (PMDK, libnuma)
- ⚡ Language support does not expand to legacy programs (NV-Heaps)

→ Operating system support for transparent memory placement

Problems we want to tackle

- Fast-tier memory is rare
  → it must be distributed efficiently
- Process workloads shift over time
  - in intensity
  - in locality
- Memory placement decisions should be adaptable



→ Detailed runtime information on memory utilisation is necessary

Identifying which memory works best for a given process?

Conventional approaches

- Instrumentalisation
  → expensive, very accurate

Identifying which memory works best for a given process?

Conventional approaches

- Instrumentalisation
  → expensive, very accurate
- Page table scanning/manipulation
  → expensive, coarse granularity

Identifying which memory works best for a given process?

Conventional approaches

- Instrumentalisation
  → expensive, very accurate
- Page table scanning/manipulation
  → expensive, coarse granularity
- Sampling (PEBS)
  → low overhead, acceptable accuracy

Identifying which memory works best for a given process?

Conventional approaches

- Instrumentalisation
  → expensive, very accurate
- Page table scanning/manipulation
  → expensive, coarse granularity
- Sampling (PEBS)
  → low overhead, acceptable accuracy

Page Modification Logging (PML)

- Virtualisation extension for VM checkpointing, VM migration
- Hardware-based logging of write accesses
- Only works within virtualisation

Identifying which memory works best for a given process?

Conventional approaches

- Instrumentalisation
  → expensive, very accurate
- Page table scanning/manipulation
  → expensive, coarse granularity
- Sampling (PEBS)
  → low overhead, acceptable accuracy

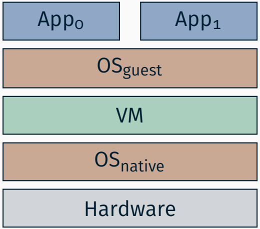
Page Modification Logging (PML)

- Virtualisation extension for VM checkpointing, VM migration
- Hardware-based logging of write accesses
- Only works within virtualisation

→ Is virtualisation viable for gathering memory access statistics?

vmmload: A minimal virtualisation layer

- Guest OS introduces overhead
- Processes should communicate with the host OS
- Processes should communicate with other processes

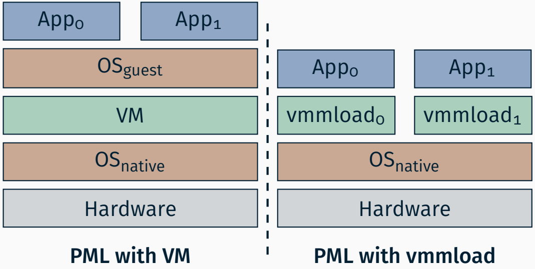


PML with VM

vmmload: A minimal virtualisation layer

- Guest OS introduces overhead
- Processes should communicate with the host OS
- Processes should communicate with other processes

→ **vmmload** as minimal hypervisor



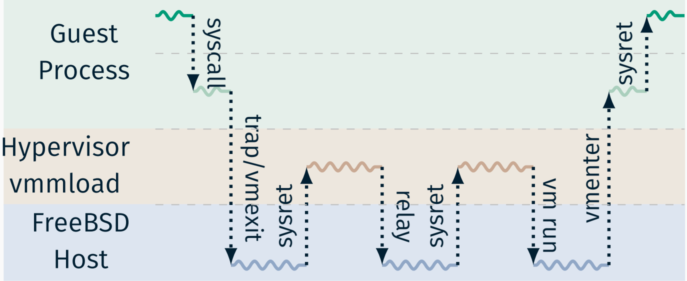
Relaying of System Calls

Requirements

- Isolate hypervisor from guest
- Interface with host OS

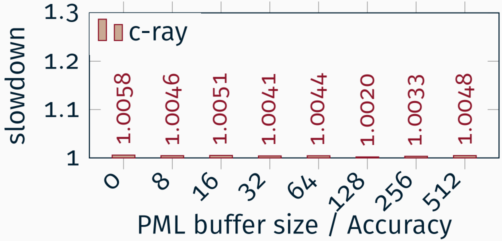
Implications

- Emulate system calls that manipulate the issuers process' state
- Translation of memory addresses



Slowdown

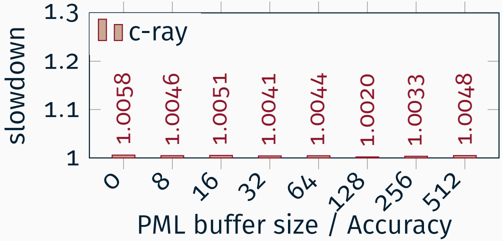
CPU-bound processes



- CPU-bound processes are hardly affected
- Delay incurred by greater PML buffer size is low

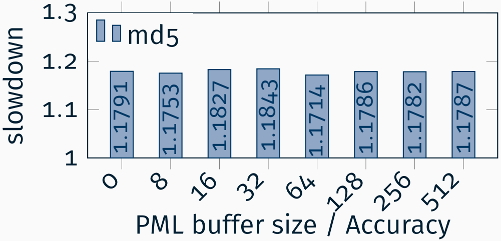
Slowdown

CPU-bound processes



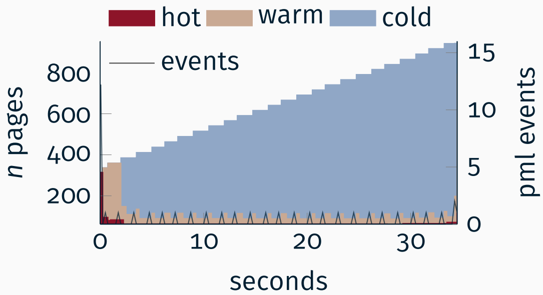
- CPU-bound processes are hardly affected
- Delay incurred by greater PML buffer size is low

I/O bound processes



- Highly interactive processes
- System call overhead introduces great slowdown

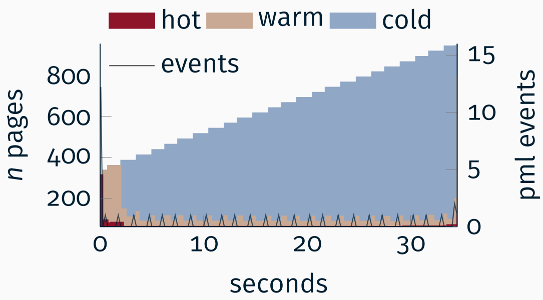
Access Patterns for *c-ray*

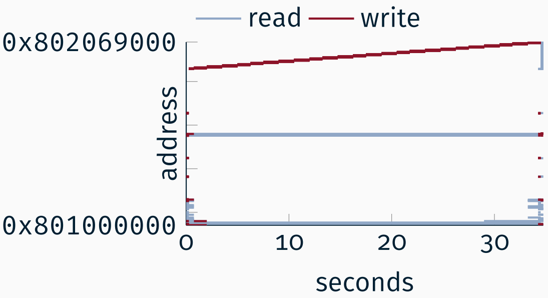


Temperature

- Identify total number of pages
- Identify frequently accessed pages
- Frequency of PML events

Access Patterns for *c-ray*





Temperature

- Identify total number of pages
- Identify frequently accessed pages
- Frequency of PML events

Distribution

- Distinguish between *read*/*write*
- Alteration of working set
- Sparsity/density of accesses

Summary

We have

- ✓ shown that **vmmload** can collect memory access statistics
- ✓ achieved statistics over read/write accesses

Next, we plan to

- 🕓 Reduce system call overhead to $\frac{1}{4}$ with kernel integration
- 🕓 Derive memory placement/migration decisions

→ Source code is freely available

Summary

We have

- ✓ shown that **vmmload** can collect memory access statistics
- ✓ achieved statistics over read/write accesses

Next, we plan to

- 🕓 Reduce system call overhead to $\frac{1}{4}$ with kernel integration
- 🕓 Derive memory placement/migration decisions

→ Source code is freely available

→ Thank you for your attention