HawkEye: Efficient Fine-grained OS Support for Huge Pages

Ashish Panwar[1], Sorav Bansal[2], K. Gopinath[1]

[1]Indian Institute of Science, [2]Indian Institute of Technology Delhi

OS Challenges

Huge pages naturally induce:

- High allocation latency (page zeroing)
- Memory bloat (internal fragmentation)
- Perf constraints due to fragmentation [2]
- Fairness challenges in resource allocation

Fundamental conflicts across optimizations:

- Memory bloat vs. performance
- Latency vs. the number of page faults

HawkEye

Data driven approach for automated operating system support for huge pages.

Key optimizations [2]:

- Asynchronous page pre-zeroing
- Content deduplication based bloat recovery
- Access pattern based (fine-grained) allocation
- Fairness driven by performance counters

Resolves fundamental conflicts!

Background and Motivation

Bloat vs. performance: Partially used VA regions.

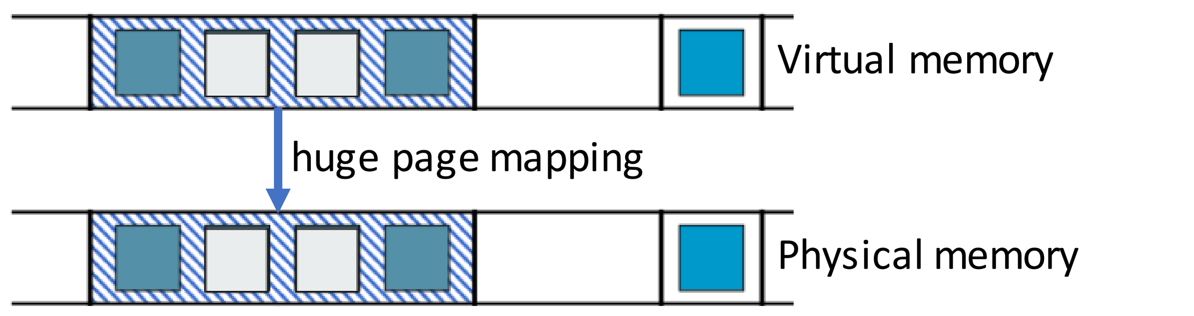

Figure 1: Internal fragmentation

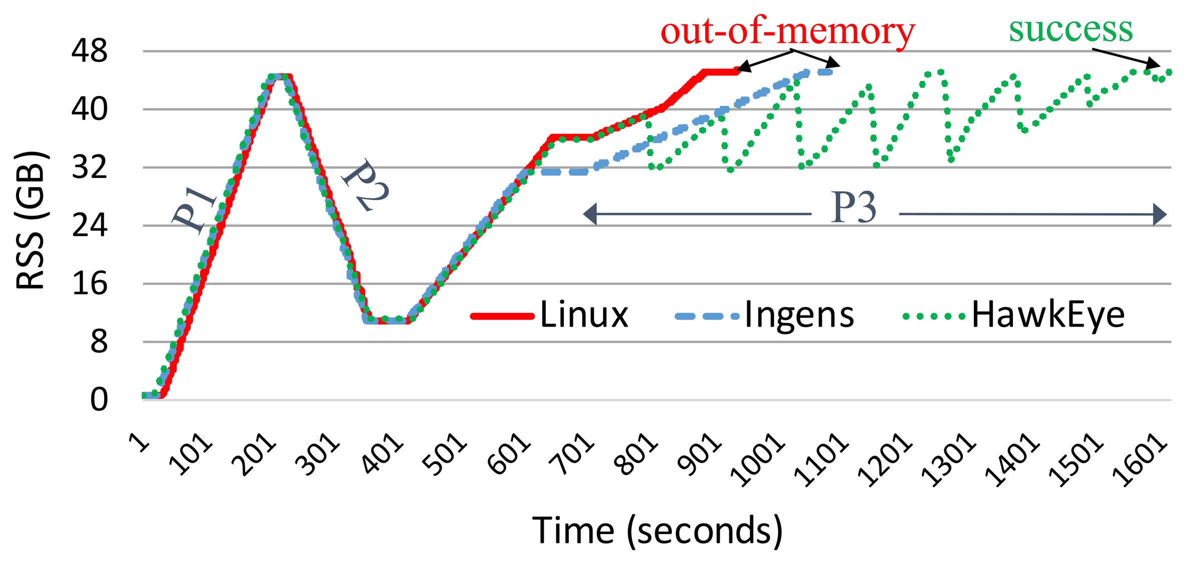

Figure 2: Resident Set Size (RSS) of Redis server across 3 phases: P1 (insert), P2(delete) and P3(insert).

- Manual tuning is a hard problem.
- Sub-optimal settings risk out-of-memory!

Synchronous vs. asynchronous

- Synchronous huge page allocation (Linux THP): high performance and high bloat
- Utilization threshold-based allocation (Ingens [1]): tunable bloat vs. performance

Latency vs. the number of page faults: Page zeroing contributes to high allocation latency.

Current state-of-the-art:

- Synchronous allocation (Linux THP): during page fault (high latency, fewer page faults)
- Asynchronous allocation (Ingens): in the background (low latency, high number of page faults)

Hard to get the best of both worlds!

Memory fragmentation External fragmentation limits huge page allocations.

Defragment memory and promote huge pages in the background.

Maximizing performance with limited contiguity:

- **Key**: Identify most profitable candidates
- Coarse-grained: inter-process selection (important for fair distribution of memory contiguity)
- Fine-grained: intra-process selection

- Current systems favor opposite ends of the design spectrum for tradeoffs involved in OS-based huge page management
- HawkEye breaks the fundamental tension with adaptive policies based on runtime characteristics of the system

Low Memory Pressure	High Memory Pressure
1. Fewer Page Faults	1. High Memory Efficiency
2. Low-latency Allocation	2. Efficient Huge Page Promotion
3. High Performance	3. Fairness

Figure 3: Ideal OS design objectives

Design and Implementation

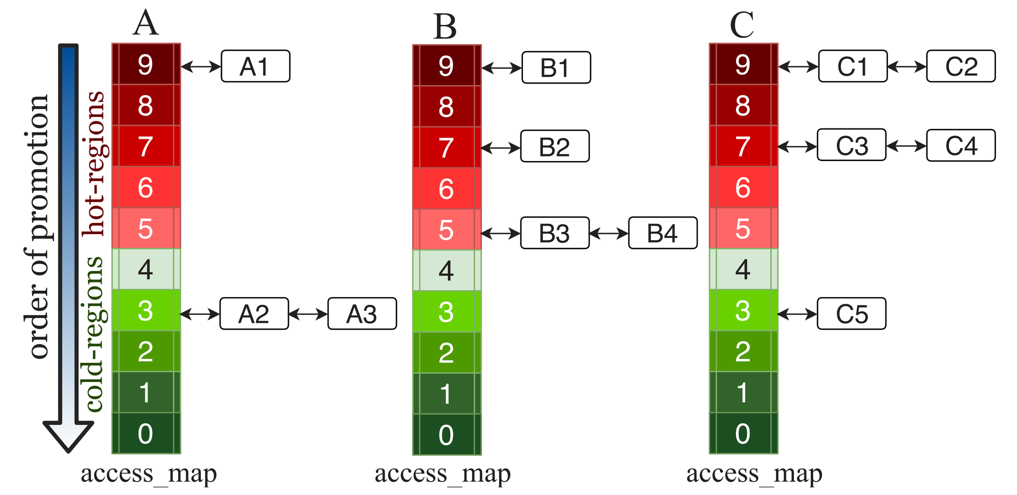

Figure 4: A sample representation of `access_map`

Dealing with latency:

- Pages zero-filled in the background
- Non-temporal writes (avoid cache pollution)
- Both **aggressive** & **low latency** allocation
- What about memory bloat?

Dealing with bloat:

- Unused base pages remain zero-filled
- Scan to detect zero-filled allocations
- Typically scanning a few bytes is enough
- Dedup zero-filled pages (same page merging)
- **Automated bloat vs. perf management**

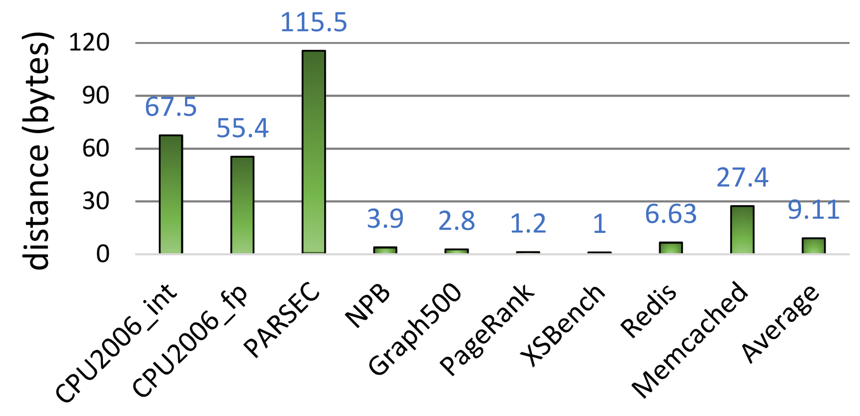

Figure 5: Avg dist to first non-zero byte in 4KB pages

Fine-grained intra-process allocation:

- Crucial under memory fragmentation
- Periodic page table access-bit tracking
- **access-coverage:** # base pages accessed per sec (profitability index of huge page promotion)
- **access_map:** Prioritize (arrange) promotion candidates based on `access_coverage`
- Yields higher profit per huge page allocation

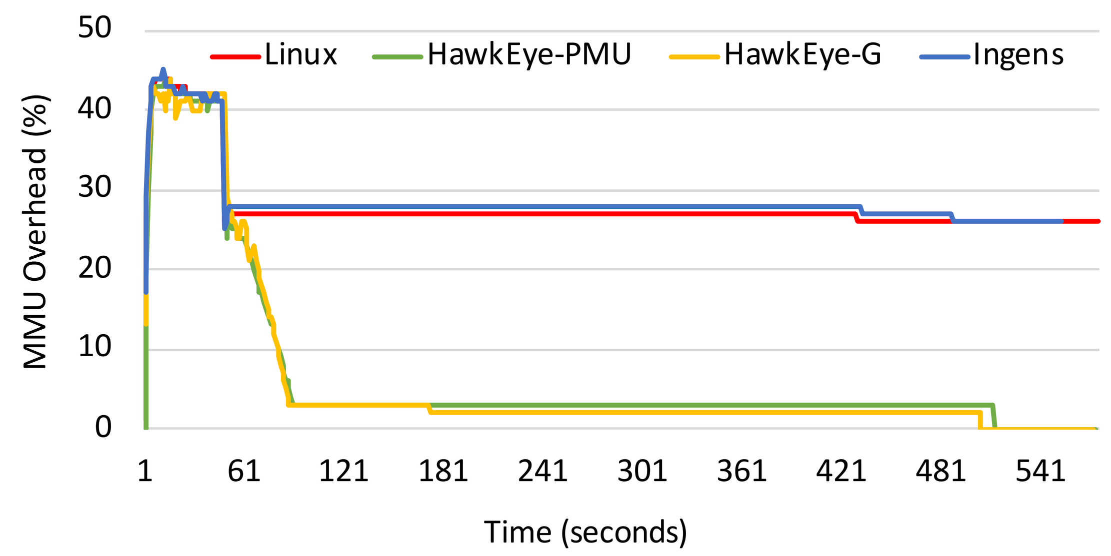

Figure 6: MMU overhead over time for XSBench

Fair inter-process allocation:

- Identifying sensitivity: Profile hardware performance counters (low cost, precise!)
- Treat MMU overhead as a system overhead
- Policy: Distribute MMU overhead equally
- Implementation: Prioritize promotion for the process with highest MMU overhead (HawkEye-PMU)
- Generalized version based on access_map alone (HawkEye-G), important for VMs

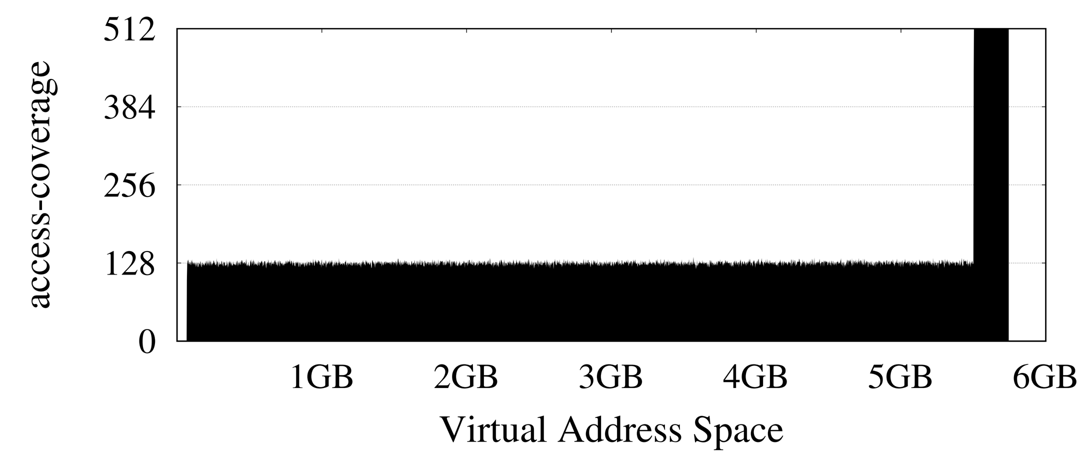

Figure 7: Access-coverage across XSBench VA space

Result Highlights

- 14× faster VM initialization
- 1.26× higher throughput (Redis PUTs)
- Up to 44× higher profit per promotion
- 5%–50% performance improvement in bare-metal (even higher under virtualization)
- Compliments memory ballooning

References

[1] "Coordinated and Efficient Huge Page Management with Ingens", Y. Kwon, H. Yu, S. Peter, C. J. Rossbach and E. Witchel. OSDI 2016.

[2] "Making Huge Pages Actually Useful", A. Panwar, A. Prasad, K. Gopinath, ASPLOS 2018.

[3] HawkEye source is available at: https://github.com/apanwariisc/HawkEye