Rebooting Virtual Memory with Midgard

Siddharth Gupta, Atri Bhattacharyya, Yunho Oh‡, Abhishek Bhattacharjee†, Babak Falsafi, Mathias Payer

EcoCloud, EPFL §Sungkyunkwan University †Yale University

Virtual Memory Performance Pitfall

Increasing memory capacity puts pressure on TLBs
- Servers feature TB-scale memory hierarchies [AWS, GCloud]
- TLBs provide only MB-scale coverage with thousands of entries/core
- Frequent, long-latency page table walks are a performance bottleneck

But huge pages are not a panacea
- Huge pages increase memory capacity and require physical contiguity
- But cause memory fragmentation by introducing multiple page sizes

Virtual Memory Areas (VMAs)

VMAs represent data sections present in address spaces
- Apps organize their address space into VMAs (e.g., heap, stack, code)
- Each VMA is divided and mapped to physical pages

Access control should be performed using the default VMAs
- Permissions are inherently defined at VMA granularity
- Need an intermediate address space to resolve synonyms!

Midgard as an intermediate address space
- Logical and global address space managed by the OS
- Deduplicated VMAs are mapped to Midgard and then to pages

Placement of address spaces in the memory hierarchy
- Perform access control and cache hierarchy lookups at VMA granularity
- Perform memory management using page granularity

System Design

Frontside translation (VA → MA)
- Only 10s of VMAs per thread as working set
- Small and fast VLB per core with easy refills from a small VMA table

Backside translation (MA → PA)
- Optimized page table walks using in-cache address translation
- Optional MLBs co-located with memory controllers for spatial locality
- Stock cache coherence protocol controls PTE copies (w/o MLBs)

Data Contiguity Can Help

POSIX VMAs are the inherent source of data contiguity

Evaluation

Baseline overhead increases with cache hierarchy capacity
- Midgard enables the overhead to scale with the cache hierarchy capacity

Midgard future proofs VM by introducing VMAs in hardware