NUMA and VM Scalability

Mark Johnston
markj@FreeBSD.org

FreeBSD

FreeBSD Developer Summit
MeetBSD 2018
October 18, 2018
Non-Uniform Memory Access

Motivation
- Scalable multiprocessing
- Target commodity systems

Assumptions
- CPU caches are coherent
- Small number of NUMA domains (usually 2 or 4)
- Low NUMA factor (20-50%)
- NUMA domains are balanced
OS Goals

- Balance resource (memory controller) utilization
- Sane default NUMA allocation policies
- Allow applications to declare intent
- DTRT for static allocations (per-CPU data, DMA, etc.)
- Handle local memory shortages gracefully
OS Support

NUMA awareness:
- CPU scheduler
- cpuset(2)
- busdma(9)
- Memory allocators: UMA, malloc(9), kmem_malloc(9), kstacks, etc.

SMP scalability:
- Page allocator
- Page queues
- Buffer cache
SRAT parser and `vm_phys` domain awareness
- r210550, r210552 (2010)
- First-touch allocation policy, useful with CPU pinning
- Changed to round-robin in r250601 (2013)

Per-domain page queues
- r254065 (2013)

projects/numa (2014)

VM_NUMA_ALLOC, `numactl(8)`
- r285387 (2015)
- First attempt at user-configurable policies
- Included a SLIT parser, currently not used by the kernel
NUMA/Scalability project

- 2017/2018, many commits
- Work by Jeff Roberson, sponsored by Limelight, Netflix, Isilon
- Plumb int domain through various layers
- Define NUMA allocation policy abstraction
- Provide userland interface for specifying allocation policy
- Address VM and buffer cache bottlenecks
Structure defining a domain selection policy

Immutable

Iterator state is defined externally (struct domainset_ref)
  - Contains a pointer to a domainset
  - Embedded in struct thread and vm_object_t

`vm_domainset_*()` applies a domainset to an iterator

Can restrict to a subset of system’s domains

Some predefined policies can be used
  - DOMAINSET_PREF(1): “Allocate from domain 1 or fall back”
  - DOMAINSET_RR(): Global round-robin
domainset(9) policies

DOMAINSET_POLICY_ROUNDROBIN

- Cycles through domains: \( d = \text{iter++ \% ds->ds_cnt} \)
- 0, 1, 2, 3, 0, 1, 2, 3, 0, ...

DOMAINSET_POLICY_FIRSTTOUCH

- Pick the domain of the current CPU: \( d = \text{PCPU_GET(domain)} \)

DOMAINSET_POLICY_PREFER

- Pick the domain specified in the policy: \( d = \text{ds->ds_prefer} \)
- Fall back to round-robin when free pages are scarce

DOMAINSET_POLICY_INTERLEAVE

- Domain is a function of the \( \text{pindex} \)
- Round-robin with a stride, for successive indices
- 0, 0, ..., 0, 1, 1, ..., 1, 0, 0, ...
- Superpage-friendly: use a stride of 512
vm_domainset

vm_domainset_iter_page_init(&di, obj, pindex, &domain, &flags);
do {
    m = vm_page_alloc_domain(obj, pindex, domain, flags);
    if (m != NULL)
        break;
} while (vm_domainset_iter_page(&di, obj, &domain) == 0);

return (m);
Userland interface

- Domain selection policies integrated into `cpuset(1)`
- Each `cpuset` has an associated `struct domainset`
- Allows specification of a policy for a thread, process, jail
  - `cpuset -n rr:0,2 make buildworld`
  - `cpuset -g -s 0`
- `cpuset_getdomain(2), cpuset_setdomain(2)`
- Userland threads default to first-touch
  - Domain selection overridden to preserve superpage reservations
Memory allocators (1)

UMA, malloc(9)
  ▶ No policy at the caching layer (fast path)
  ▶ Default round-robin policy at the slab layer (zone iterator)
  ▶ UMA zone policy: UMA_ZONE_NUMA for first-touch
  ▶ uma_zalloc_domain(2), malloc_domain(2)

kmem_malloc(9) and friends
  ▶ Round-robin policy (thread iterator)
  ▶ Multiple vmem(9) arenas provide striping for superpages

busdma(9)
  ▶ Bus can be queried for domain affinity (_PXM method)
  ▶ DMA tags cache local domain index
  ▶ DMA allocations use malloc_domain(9) with local domain
Memory allocators (2)

vm_page_alloc() and friends

- Source of user memory allocations (page faults, etc.)
- Not always under user control (e.g., libc.so)
- Policy specified by VM object (may be absent), or thread
- vm_page_alloc_domain()

Kernel stacks

- Global round-robin policy (thread iterator)
- Kernel stacks are cached
- We can do better (e.g., ithread kstacks)
Low memory handling

- Each domain has page queues, page daemon, laundry thread
- Page domains are mostly independent
  - Per-domain free page targets, laundry targets
  - OOM kills occur only when all domains are depleted
  - Does not work well if most of a domain is wired (e.g., by ARC)
- `vm_wait_doms()`: sleep until one of the specified domains has some free pages
Scalability improvements

- PID controller for free page target
- Split free page mutex and add per-CPU free page cache
- Fine-grained reservation locking
- Lockless page daemon wakeups and v_free_count updates
- Per-CPU v_wire_count accounting
- Page queue batching
- Lazy dequeue of wired pages
- Buffer cache sharding, locking improvements
Future Work

NUMA:
- Non-x86 support (arm64 and powerpc64)
- Statistics collection
- libnuma, msetdomain(2)
- Static allocations (pcpu(9), kernel thread stacks, etc.)
- More affinity plumbing (per-mountpoint policy?)
- ZFS integration
- taskqueue(9) integration

Scalability:
- Split user (mlock(2)) and kernel wired page accounting
- Lockless per-page queue state
- Lockless vm_page_hold()
- Improve PQ_ACTIVE scalability in the page fault handler