Status of the Linux Slab Allocators

David Rientjes
rientjes@google.com

SCALE 9X

February 26, 2011
Los Angeles, California

Google

Status of the Linux Slab Allocators

- As of 2.6.37.1, the latest stable release kernel
- When configuring a kernel.org kernel, few users can make an informed decision on which slab allocator use
- `defconfig` is of little help, `CONFIG_SLUB` regresses significantly on some workloads
- Requires a rebuild to change
- *How do you choose which slab allocator to use?*
- *For systems running a consistent set of workloads, how do you determine what is in your best interest?*
- *For systems running a wide variety of workloads, what trade-offs do you have to make when making a decision?*

SLUB

- Unqueued slab allocator, enabled in `defconfig`
- Merges slab caches with same properties together for cpu cache optimizations
- Object allocation from cpu slab (fastpath), larger orders help
- If full, fallback to per-node partial lists with per-node locking
- Ordering of partial list can cause “*slab thrashing*” depending on allocation and freeing pattern
- Worst-case: allocate new slab from the page allocator, requires a fast page allocator
- Object free to cpu slab (fastpath)
- Otherwise, free to full or partial slab and update partial lists as necessary
- When slab is empty, it can be freed back to the buddy allocator unless it should stay on the partial list

SLUB Debugging

- Much superior debugging support to any other allocator
- Can be enabled per slab cache rather than globally
- No kernel rebuild is necessary, only reboot to activate via command line or `sysfs` interface
- **Poisoning:** detects use-after-free
- **Red zoning:** detects use before and after object
- **User tracking:** stores alloc and free caller
- **Tracing:** emits full stack on alloc and free to the kernel log (very verbose)
- Slab cache merging may spew too much information or obfuscate the cause of a problem (may be disabled)
- May increase the slab order to increase as a result (may be disabled)

SLAB

- Deprecated, very little development
- Many distributions still ship with `CONFIG_SLAB` even though it is not the kernel default
- Object allocation from array of free objects (fastpath)
- Otherwise, refill array with shared objects that have same memory affinity, if possible
- If not, allocate object from partial slabs and fallback to the page allocator if necessary
- Object free returns objects with affinity to local array, otherwise frees alien cache
- Respects thread's mempolicy and attempts to allocate from correct node, if possible
- Cache reaper runs every few seconds and attempts to clear the per-cpu caches and free empty slab, if possible

SLOB

- "Simple List of Blocks" heap allocator with alignment and NUMA support
- Very small memory footprint (less than a page of text)
- Slab pages organized into linked-list of free blocks
- Separate lists depending on object size (<256 bytes, <1024 bytes, and all others) to reduce fragmentation
- Object allocation done by allocating first free set of blocks in slab list
- No optimizations for cpu cache: allocation is done by address, not queue
- Often used in embedded devices or machines with strict RAM limitations

Tools for debugging and development

- **failslab:** fault injection for `kmalloc()`, `kmem_cache_alloc()` to test error handling of new code
- **kmemcheck:** checks for use of uninitialized memory, requires slab hooks for initialization
- **kmemleak:** scans through memory and emits the number of unreferenced objects, check debugfs file for list

Kernbench

- Benchmarks cpu throughput using a kernel build

16-core machine with 32GB memory (4 nodes)
10 iterations

	Half load		Average load		Maximal load	
	SLAB	SLUB	SLAB	SLUB	SLAB	SLUB
Elapsed time	9.960	9.864 *(-1.0%)*	6.197	5.998 *(-3.2%)*	6.266	6.281
User time	48.190	48.042	48.828	48.610	49.807	49.586
System time	4.773	4.627 *(-3.1%)*	4.885	4.754 *(-2.1%)*	4.913	4.783 *(-2.6%)*
Percent cpu	531.50	533.70	705.90	717.55 *(+1.7%)*	772.10	777.37
Context switches	260.10	235.30 *(-9.5%)*	1670.35	1550.10 *(-7.2%)*	5224.37	5090.53 *(-2.6%)*
Sleeps	8601.30	8521.40 *(-1.0%)*	8895.80	8920.30	8437.73	8449.30

Google

Netperf TCP_RR

- Benchmarks round-robin networking performance

16-core machines with 32GB memory (4 nodes) each
10 iterations, 60 seconds each

Threads	SLAB	SLUB
16	129667	116138 *(-10.4%)*
32	136506	120057 *(-12.1%)*
48	141470	125291 *(-11.4%)*
64	147653	131053 *(-11.2%)*
80	154212	134125 *(-13.0%)*
96	153331	134216 *(-12.5%)*
112	163065	134725 *(-17.4%)*
128	158108	136577 *(-13.6%)*
144	161774	144855 *(-10.5%)*
160	167896	151248 *(-10.0%)*

Google

SLQB

- Queued allocator, initially proposed for systems not benefiting from SLUB
- Contains much of the core infrastructure of SLUB
- Object allocation from per-cpu freelist, minimizes cacheline bouncing and returns hot objects (fastpath), otherwise fallback to partial list
- Freelist has a watermark that, when passed, flushes free objects back to slab allowing them to be freed to the buddy allocator if empty
- Objects that are freed on different cpus on which they were allocated are flushed to a remote freelist that eventually move back to the allocating cpu
- Locking to reach into remotely freed lists is controlled by batching and watermarks
- Currently abandoned, may resurface

SLUB+Q

- Effort to unify the best qualities of both SLAB and SLUB using the queues from the former and infrastructure from the latter, such as debugging
- Essentially makes the “unqueued” SLUB use a cpu queue
- Adds an object bitmap within the page struct for management, can cause slight increase in memory usage
- Does not do cache reaping like SLAB, fully controlled by page reclaim
- Respects mempolicies of allocating task on a per-object level
- Adds shared and alien caches for cross cpu allocations
- Still regresses for large machines on some benchmarks
- Little recent development

SLAM

- Mutable slab allocation system
- Intended as a drop-in replacement for all slab allocators (ambitious)
- Predicated on the concept that all slab caches do not behave the same
- Configurable per cache per cpu behavior depending on its usage and memory requirements
- Single allocation fastpath, different slowpaths depending on allocation and free patterns
- Advisable cache behavior within the kernel with `kmem_cache_advise()`
- Adjustable via `sysfs`, including automatic flight
- Currently being developed

Google

Slab allocation development

- Subsystem co-maintainers
  - Pekka Enberg <penberg@cs.helsinki.fi>
  - Christoph Lameter <cl@linux.com>
  - Matt Mackall <mpm@selenic.com>
- Additional development
  - David Rientjes <rientjes@google.com>
  - Nick Piggin <npiggin@suse.de>
- linux-mm@kvack.org mailing list, LKML