Memory Management under Linux: Issues in Linux VM development

Christoph Lameter, Ph.D.
Technical Lead, Linux Kernel Software
Silicon Graphics Inc.
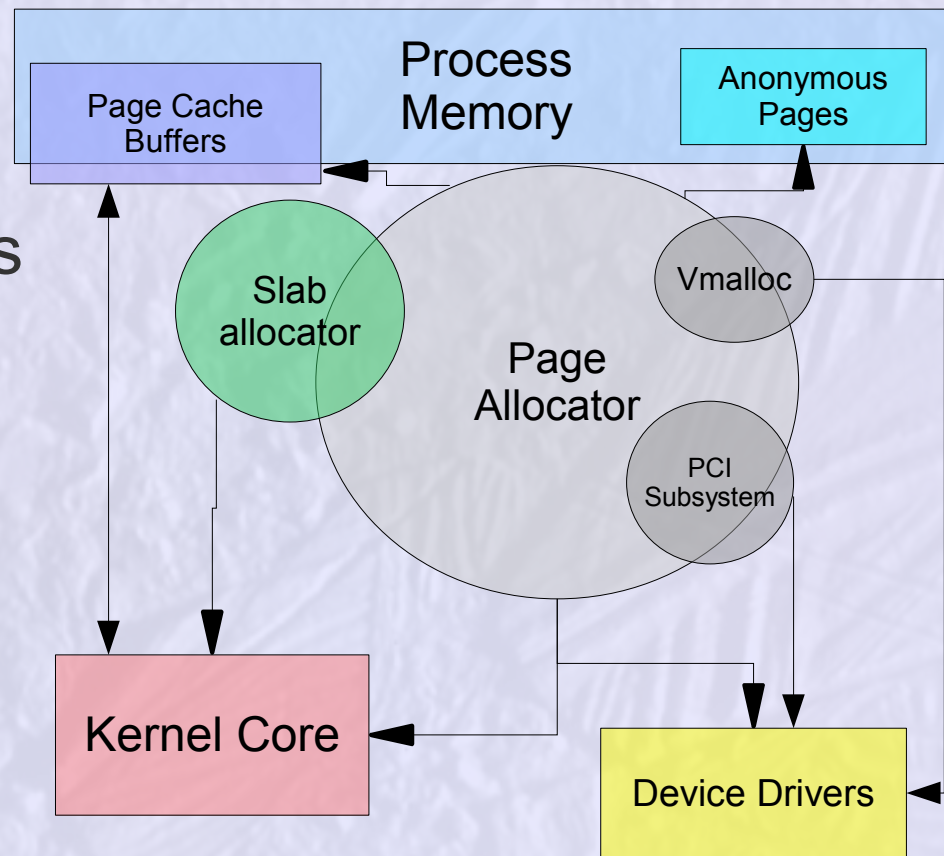clameter@sgi.com

2008-03-12

- Short overview of Linux Memory Management
- Projects that were merged in the last year
- Problems with solutions
- Unsolved Problems
- Future Directions

- Virtual Memory Management
  - Provide memory to the processes and for the operating system services
  - A significant influence on performance.
  - Includes various forms of synchronization in Multiprocessor systems
- Resource management
  - Allocate memory so that the processes competing for memory make the best progress.
- Linux challenges
  - Increasing complexity of the VM as the number of processors and the memory sizes grow
  - Management of memory in 4k chunks in a world with machines that have Gigabytes of memory.

Memory Management Components

- 4k Chunks of Memory
- Page allocator
- Higher order allocations
- Anonymous memory
- File backed memory
- Swapping
- Slab allocator
- Device DMA allocator
- Page Cache
- read() / write()
- Mmapped I/O.

Process Memory

Page Cache Buffers

Anonymous Pages

Slab allocator

Vmalloc

Page Allocator

PCI Subsystem

Kernel Core

Device Drivers

Philosophy of Linux Memory Management

- All memory must be put to use. Unused memory is a waste of memory.
- Memory is freed if we find another use for it. It is not freed without reason.
- Memory is managed in "zones". A zone is a range of memory suitable for a certain purpose.
  - Memory suitable for legacy I/O devices
  - Memory that is easily reclaimable
  - Memory that requires explicit handling (HIGHMEM)
  - General memory
- NUMA implemented by adding more zones.
- A major problem of memory management under NUMA is selecting the zone from which to get memory for a process.

- Anti-Fragmentation logic for the page allocator (Mel Gorman)
- Memoryless node support
- Device based dirty throttling
- SPARSEMEM Virtually mapped memmap
- Quicklists
- 64k support on IA64 and in the ext2/3/4 filesystems.
- Cgroups – a framework for resource control.
- Support for more processors on x86_64
- Reducing per cpu allocation overhead overhead (support for up to 4k/16k cpus on x86_64)
- Reducing stack use for cpumasks
- SLUB

- Current waste of memory. A device must have enough pages buffered up to make it work at optimal speed. Pages should be kept hot

- Devices may not run at full speed because dirty pages may be limited due to another slow device creating dirty pages

- New dirty throttling algorithm calculates the dirty rate of a process and the write out rate of the device and then makes the process produce dirty pages at the rate of disk I/O

- More memory available. Higher device speed. Some report doubling of speed in some situation

- Addresses most of our dirty pages issues. Not sure though if it addresses cases where a node ends up full of dirty pages

- Dirty pages are only limited on the system as a whole
- Small cpusets can have all of memory dirty
- No memory is reclaimable thus we get an out of memory errors (and thoughts of adding memory come up).
- Changes the calculation of dirty ratios based on the number of pages in a cpuset.
- Provides an upper limit in addition to the limits imposed by the device based throttling.

- Old allocator SLAB
  - Exponentially increasing memory use the more nodes and processors.
  - Per object NUMA control which led to inefficiencies in the alloc and free handler. Bugs still surface.
  - No ability to defragment memory
- New allocator SLUB:
  - Memory efficient (->low cache footprint -> more speed)
  - Speed through uses of atomic operations not queuing objects
  - Framework for slab reclaim
  - Support for higher order allocations to increase speed
  - Simpler and easy to understand.
  - cmpxchg_local fastpath that does not require disabling interrupts. Cycle could is reduced by 30%-60%

- Page allocator dependency
- Variable order slabs
- Tunable slabs
- Cmpxchg_local fastpath
  - 30-60% performance increases
  - Can be used in combination with tunable orders
- Slab defragmentation

- Addresses High TLB fault cost on IA64.
- Allows larger file system buffers.
- Also scales page allocator.
- Increases addressable memory for 3 level page tables.
- Enables very large systems. 64K page size seems to be a requirement for systems with 2k or 4k processors and about 1k nodes.

- Included in 2.6.24.
- Per node status information
- SLAB issues. Need per node structures for nodes that have no memory.
- Introduction of a node status
  - Node is possible
  - Node is online
  - Node can accept high memory allocations
  - Node has normal memory
  - Node has processors

- SMP mechanism to control memory
- Adds a per cgroup LRU which establishes page ownership to the cgroup.
- Potential issue if it is to replace cpusets: No per node LRUs!
- Cpuset/cgroup kill on swap?

- Some performance improvement vs. DISCONTIG and regular SPARSEMEM.

- Simplifies code in the kernel in general

- Allows SPARSEMEM to avoid using page flags for section Ids.

- x86_64 has VMEMMAP as the only memory model. Discontig / Flatmem removed.

- Same is to be done for IA64. Tony Luck indicated that he wants to do this.

- i386: Key problem to freeing page flags here with NUMAQ. Problem is that SPARSEMEM_STATIC leaves no available page flags.

- Future: Movable 2M pages, fallback to 4K?

Problems for which patch sets and solutions exist

- Off lining memory and nodes: Memory unplug
- Performance and scaling problems because of 4k page size in large systems or due to page pinning
  - LRU optimizations (Rik van Riel)
  - Not scanning pinned, mlocked or anonymous pages (Rik)
  - Large Blocksize patchset (64k blocksize)
- Too many dirty pages accumulate on some nodes
  - Cpuset/cgroup based dirty throttling
- Swap is not acceptable for HPC applications
  - Cpuset/cgroup kill on swap (Paul Jackson)
- Support for more processors (4k – 16k - 64k)
  - cpu_alloc
  - cpumask size reduction work (Mike Travis)

- Almost empty slabs can consume lots of memory
  - Slab defragmentation
- Unreliability of higher order allocations
  - Fallback to order 0 in SLUB
  - Virtual Compound pages (for stacks etc)
- Avoiding cacheline contention in percpu allocation
  - cpu_alloc
- Effective per cpu counter and other operations
  - cpu_ops
- DMA zone problems (OOM issues, NUMA memory balancing)
  - DMA Zone allocator (Andi Kleen)
  - Getting rid of ZONE_DMA(32?)
- Lack of page flags (removal of SPARSE_STATIC)

- Page allocator performance for order 0 pages
- Page allocator performance for higher order pages.
- Devices or subsystems pinning memory (MMU notifier, EMM notifier exist but are unable to sleep without additional modifications).
- Support for I/O from vmalloc'ed memory
- Removal of SPARSEMEM_STATIC from i386 to free up page flags.

- Pinned pages are not reclaimable (RDMA, Xpmem etc)
- Too many per node may lead to Out of memory errors
- A too large percentage will lead the VM to continually scan through pages that are not reclaimable
- Live locking under load.
- Solution is to remove the pages from the reclaim list and go to other nodes if there is no memory available.
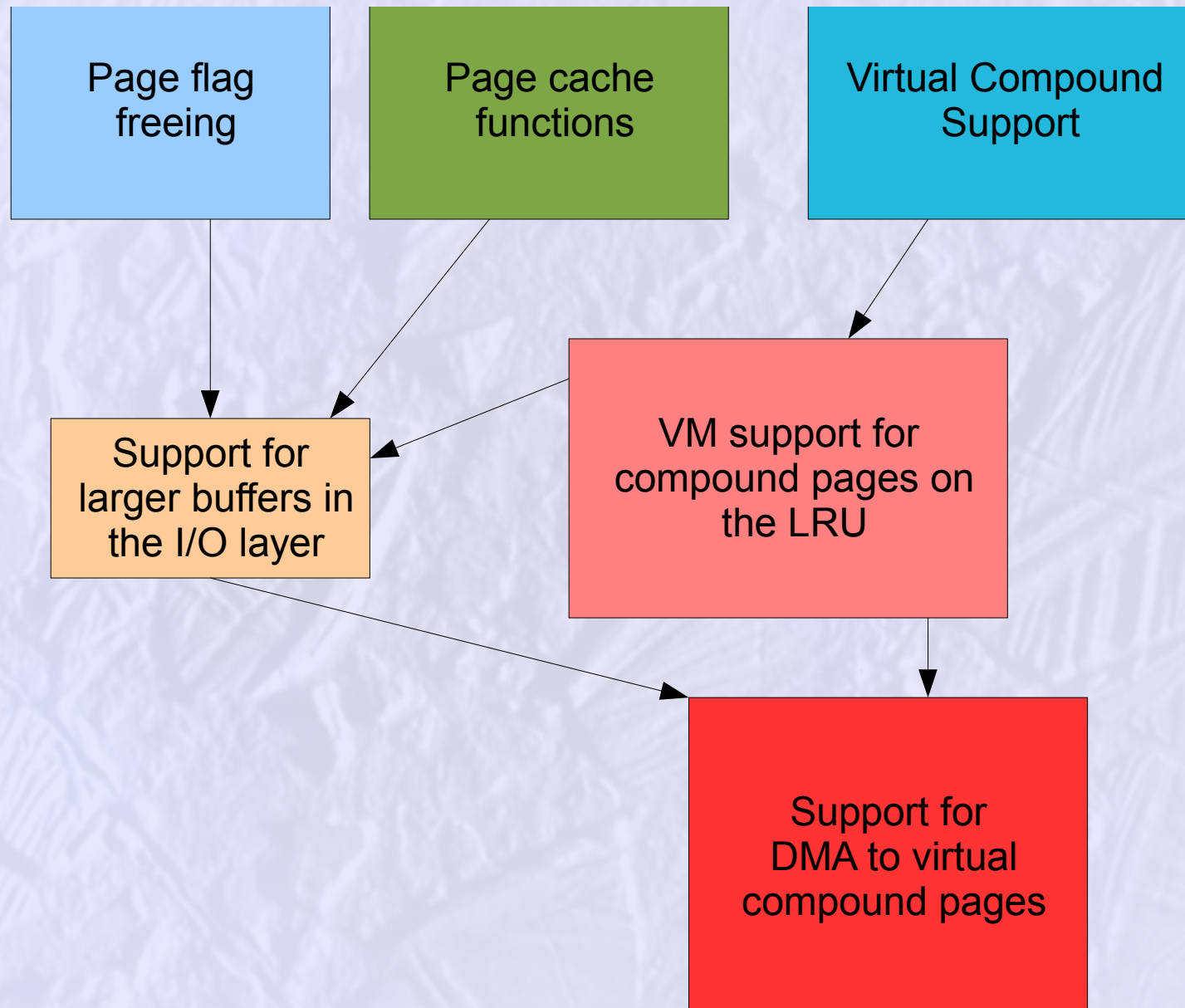- Work in progress by Rik van Riel.

- Avoids vmalloc. The use of virtually mapped memory needs page table lookups and 4k ptes. Vmalloc is only used if there is no contiguous memory available.

- Reduces TLB pressure

- Allow the use of antifrag measures to speed up the system.

- Allows large stack sizes

- Avoids failure on order 1 stack allocations

- Provides base for higher order page cache fall back feature.

- Allows to provide fall back for various higher order allocations in the kernel (as f.e. useful for node plug in)

- Allows reliable use of higher order allocations.

- Needed for scaling VM and I/O
- Avoids the LRU scaling problem
- Scales I/O. Reduces scatter / gather list sizes. Linear I/O for large amount of memory possible.
- Potential to make huge page use transparent.
- May avoid TLB pressure issues

- Fastpath is not fast. 8x slower than slab allocator fastpaths.
- May need to assign blocks to various higher orders.
- Memory defrag missing. Mel did a draft about a year ago but so far no demand. More higher order page cache use may make the defrag logic necessary.
- Easily fragments.

Patch sequence for Compound page support in the Page Cache

Page flag freeing

Page cache functions

Virtual Compound Support

Support for larger buffers in the I/O layer

VM support for compound pages on the LRU

Support for DMA to virtual compound pages

- Driver needs to directly map user space pages for DMA transfer or other special needs.
- VM thinks the pinned pages are only temporarily unavailable and continues attempts to reclaim memory.
- Works fine for small amount of pinned pages.
- The more pages are pinned the more the VM will uselessly attempt to reclaim memory.
- One solution: Take pinned pages off the LRU so that they are no longer scanned.
- The other solution: Send a notification to the device driver so that the pages are unmapped and memory can be reclaimed.

- Questions?