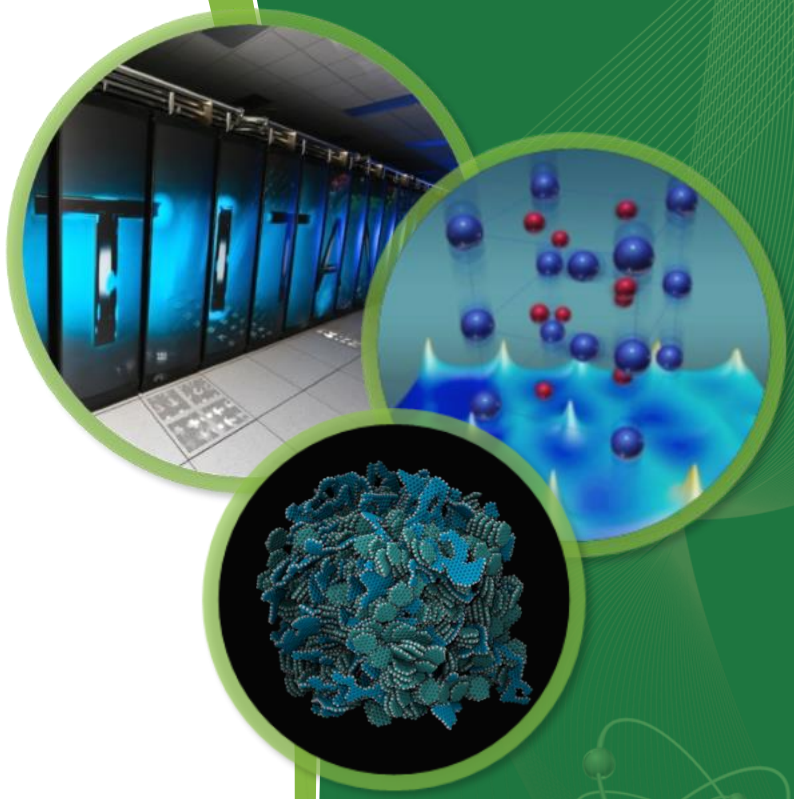
Lustre and memory

James Simmons

Oak Ridge National Laboratory



More than memory allocation

- Few understand the Lustre code
- Memory is core functionality
- With Lustre code demystified people can contribute

High level view

- User space allocations
  - Lustre utilities and tools
- Kernel space
  - NUMA aware
  - Classes
    - Pages
    - SLOB / SLUB / SLAB
      - SLOB – simple list of objects, SLUB – no object queues
    - Virtual memory manager
    - Others not used by Lustre -> CMA, iomem
- Excellent read - https://www.kernel.org/doc/gorman/pdf/understand.pdf

Slab handling

- Created to limit memory fragmentation
- /proc/slabinfo and /proc/slabtop provide info on caches
- NUMA versions exist
- Interface to manage caches
  - kmem_cache_create, kmem_cache_shrink, kmem_cache_destroy
  - Slab caches can merge
- Managing buffers from slabs
  - kmem_cache_alloc(), kmem_cache_free()
- kmalloc is wrapper to use general purpose caches of specific order size.
  - GFP_xxx flags discussed when pages are covered

Virtual contiguous memory manager

- Functions to allocate virtual memory
  - valloc, vfree, vzalloc
- Pro : does not care about physical memory fragmentation
- Con: has global spin lock --- bad for performance
  - Why use ? – kmalloc has size limitations (arch dependent, x86 128k+)
  - Will see this when Lustre striping is > 672 for pre-pfl code
  - Libcfs/LNet uses vmalloc when allocating more than 2 pages
- Kernel data structure
  - struct vm_area + struct vm_struct
- Used for memory mapping
  - Lustre llite vvp layer and llite_mmap.c manage this.

Lustre memory allocation wrapper

- Libcfs/LNET - LIBCFS_ALLOC, LIBCFS_FREE
  - Used to find memory leaks
  - lctl set_param debug+=malloc
- Lustre – OBD_ALLOC, OBD_FREE
  - Also used to find memory leaks - leak_finder.pl
  - Removed upstream
- Why they exist?
  - Left overs from when Lustre was both user land and kernel space product
  - Written before trace events / ftrace existed for debugging
  - Kernel now has excellent memleak trace tools
- Will be replaced in 2.11 with trace event / ftrace
  - Replace wrappers with direct kmalloc / vmalloc calls

Kernel page handling

- Page allocation
  - Memory Zones
    - GFP_HIGHMEM, GFP_DMA
  - Context
    - GFP_ATOMIC, GFP_KERNEL, GFP_FS
  - NUMA node id
    - alloc_page_node(..)
  - Buddy allocator
    - Power of 2 allocations
    - alloc_pages(gfp_mask, order);
    - Single page helpers - __get_free_page(..)
- Page release
  - free_page(), free_pages()

Page management

- Pages are finite in number
- LRU (least recently used) like algorithm
  - Two list, active and inactive
  - All pages except slab allocator are managed by LRU
    - Difficult to age pages for objects created by slab allocator
  - Linked to page cache handling
- Page Cache – set of data structures that contain “backed” pages
  - memory mapped files
  - block reads from fs, HW devices
  - shared memory,
  - swap

Page cache usage for typical file systems/block devices

- For file system avoid expensive reads from devices
- Pages faulted in for memory mmap file (llite vvp and llite_mmap.c)
- Stored in page hash which is hashed on the struct address_space
- Function interfaces
  - add_to_page_cache(…), delete_from_page_cache(),
  - page_cache_alloc(...), page_cache_release()
- Filesystem interfaces to page cache
  - file_read_iter, file_write_iter, file_readpage
- readahead
  - userland syscall that moves a file into page cache

Lustre is not typical

- Issues with kernel readahead
  - Lustre tries to trim readahead window which kernel can't detect.
  - When the stride exceeds backing_dev_info::ra_pages, kernel will think it is a random read.
- Does Lustre use the page cache ?
  - Yes : metadata operations, vvp layer it llite
  - No : the CLIO abstraction

Lustre metadata caching

- Normal file systems cache exact copy of raw directory pages in cache
  - Logical offset used as index to find the page
- File names are distributed across nodes based on their hash
- Hash of first directory entry is index to directory pages cached
- Hash are not unique; collisions happen but are rare

Lustre metadata implementation

- Lustre internal xxx_read_page implementation
  - Not related to struct address_space_operations readpage member
  - Layout of readdir pages : struct lu_dir_ent
  - Data is bundled together as struct lu_dirpage, as transmitted on wire.
    - This data is stored in the page cache
  - lmv_read_page() : created lu_dir_ent from transmitted data
    - If DNE striping info call lmv_read_striped_page(). Also calls mdc_read_page()
    - else call mdc_read_page() directly – fetch if not in client page_cache
    - Pages with hash collisions are discarded in mdc_read_page
- The llite metadata interaction
  - ll_iterate/ll_dir_read : request data from mdc and fills it in for end user
  - Directory statahead functionality reads metadata into memory. Like readahead
    - Special kernel thread gathers metadata information

Lustre CLIO abstraction

- CLIO abstraction
  - Base is abstraction on top of struct lu_object, struct lu_object_header, etc …
    - struct lu_* is an infrastructre to manage object life cycle and caching
    - Struct cl_device, struct cl_object, struct cl_object_header
  - struct cl_object represents file system i.e file, stripe
    - Something like a stripe is considers a sub-object. lov_subobject.c is example. Tied to parent life cycle.
    - Subsystems create abstractions on top of struct cl_object.
      - Struct lov_object, struct lov_subobject, strct osc_object.
  - struct cl_page represents a portion of file cached in memory
    - Another layer on top of struct cl_object
    - kept in the radix tree hanging off the cl_object
    - Can be owned by struct cl_io (state machine to mange high level I/O activity)

More CLIO abstraction

- struct cl_io_slice : IO state private for a layer.
  - vvp_io, lov_io, osc_io
- Others: struct cl_locks, struct cl_read_ahead
- clio’s version of a page cache - struct cl_client_cache
  - lov module : static lov_cache, osc module : cl_cache in struct client_obd
  - Track "unstable" pages : are pages pinned by the ptlrpc * layer for recovery purposes
  - LRU of cached pages
- Abstraction on top of CLIO is always defined in xxx_cl_internal.h headers.

Looking at clients layer using CLIO

- OSC
  - osc_lock.c : osc_lock -> cl_lock handling and interactive with LDLM
  - osc_object.c : osc_object -> cl_object
  - osc_page.c : osc_page -> cl_page
  - osc_io.c : osc_io -> cl_io
- LOV
  - lov_lock.c : lov_lock -> cl_lock handling and interactive with LDLM
  - lov_object.c : lov_object -> cl_object
  - lov_page.c : lov_page -> cl_page
  - Family for sub-objects (i.e stripes)
    - lov_sublock.c, lov_subobject.c, lov_subpage.c
- LLITE
  - Not the same mapping as above. llite handle VFS <-> lustre
  - Base CLIO structure – struct ll_cl_context in llite_internal.h

Lustre ll_readpage() use of CLIO

- Base CLIO structure – struct ll_cl_context in llite_internal.h
- struct address_space_operations : readpage = ll_readpage()
  - Kernel readpage() implemenation places file data into page cache
  - Handles fast read path (ll_do_fast_read)
    - Create cl_io and get ldlm lock is expensive
    - If page exist in page cache then lock already exist so just grab from memory cache
    - cl_vmpage_page() maps page cache page to a cl_page
    - Struct vvp_page is then retrieve by cl2vvp_page()
      - Struct vvp_page is abstraction on top of cl_page related to readahead.
  - Non fast read path cl_page_find() handles map page cache page <-> cl_page

Lustre readahead use of CLIO

- CLIO struct cl_read_ahead provides glue between layers
  - lov (lov_io.c) need to work together due to the impact of striping
  - osc (osc_io.c) to handle ldlm locking
  - llite (vvp_io.c) to handle potential file group locking
- Lustre llite currently provides its own readahead infrastructure
  - Most this code exist in llite rw.c as ra_xxxx functions.
  - WARNING this most likely will change for lustre 2.11
    - https://review.whamcloud.com/#/c/26469
    - New code uses struct address_space readpages() and padata to enable parallel operations

Potential future changes

- The removal of Lustre internal readahead implementation
- Take advantage of DAX. Native support for burst buffers.
- Potential changes due to upstream client leaving staging.

Closing remarks

- Thank you for listening
- Examined memory management in Lustre kernel code
- Empower people so they can consider contributing.