I/O Performance and Concurrency

BSDCan 2008
Developers Summit

Jeff Roberson
jeff@freebsd.org

Introduction

- What is this about?
- Outline
  - Current Organization
  - Problems
  - Proposed Solutions
  - Fallout

Buffer Cache Responsibilities

- Provide KVA mappings and cache
- IO initiation
- Async IO
- Delayed IO
- Throttling write pressure
- malloc backed and VM coherent IO
- LRU buffer replacement
- Filesystem swiss army knife

Buffer Cache Bolt-ons

- Clustering
- File defragmentation via clustered writes
- Invalidation and truncation
- Metadata dependencies
- Sync support via dirty vnode list
- File level cache fairness for snapshots
- Background writes

VM I/O Responsibilities

- Maintain global page targets
- Read on page-fault, write on page-out
- LRU global page replacement
- Default GETPAGES/PUTPAGES implementations
- Cluster write in vm_pageout_clean, vm_object_page_flush
- Cluster read in vm_fault_additional_pages
- Page mapping facilities (sfbufs, pmap_qenter)

syncer Responsibilities

- Enforces the upper bound on delayed IO time
- Syncs inode times
- msyncs whole filesystems
- One of 4 daemons pushing IO out with differing goals
  - pageout: global memory pressure
  - buf: available buf space
  - syncer: maximum time interval
  - softupdate: limits dependencies

Performance Problems

- Global mapping is expensive
- Requiring KVA wastes address space
- KVA requires complex defrag/allocation operations
- Splays are slow for large objects
- Multiple splays per object requires multiple lookups
- Syncer sabotages delayed I/O

Concurrency Problems

- Reads/writes serialized by f_offset
- Non-overlapping io serialized by exclusive vnode lock
- Simultaneous readers serialized by exclusive buffer lock
- vm object and bufobj use exclusive mtx lock for lists

Structural Problems

- Syncer syncs whole filesystems and individual vnodes
- vfs_cluster is cumbersome and too intimate with the filesystem
- Direct i/o private to ffs
- Poor layering
- bufs are overly complex

VM I/O Problems

- Unaware of block boundaries for replacement decisions
- Less developed/redundant clustering
- Requires mapped pbufs despite not touching the data
- putpages requires buffer cache
- Competes with buffer cache and syncer for global memory replacement and I/O decisions.

Consequences

- We have less than 1/3rd the performance of Linux on an 8core system running mysql with the MyISAM engine
- We are needlessly KVA starved on 32bit systems
- Other I/O intense applications suffer excessive CPU overhead due to poor structure
- I/O is provably less efficient and predictable
- Organic growth has left the code complex and difficult to understand

Proposed Solution

- Relax f_offset on reads
- read/write/append byte range vnode I/O lock
- Buffer cache stays for metadata and slow filesystems
- VM based read/write & clustering routines
- Unmapped i/o
- Replace the splays with radix trees
- Change the syncer responsibilities

f_offset locking

- POSIX does not mandate any synchronization for non-device files
- POLA mandates synchronization of writers
- BSD Synchronizes everything
- Only synchronize writes, allow reads to proceed concurrently
- Fix 64bit on 32bit issues

Byte range locks

- Conceptually similar to posix ranged locking
- Reads may proceed against overlapping reads
- Writes may proceed against non-overlapping writes
- Reads and writes may not proceed concurrently
- One appender at a time
- Shared vnode lock
- Allocation protected by buf locks

VM Based VREG IO

- Do regular i/o directly against pages without the buf detour
- Unify the clustering mechanism with the VM
- Buf cache becomes meta-data only
- Filesystem independent
- Must track outstanding writes for sync
- Must limit total outstanding writes
- New delayed i/o mechanism

Unmapped bios

- Remove b_data add an array of pages and an offset
- Provide routines to temporarily map whole bios for middle layers (raid, encryption, etc)
- VM already provides faster page by page mapping
- New busdma_map_bio function for drivers
- Requires lots of plumbing but simple changes

Radix trees

- Splays reorder on insert, lots of writes, no shared locking
- Splays are order 2 trees, lots of cache lines
- Radix provides a maximum depth which bounds cache misses
- Radix are compatible with shared locking
- In progress for 2008 SoC

Syncer

- Move maximum delayed IO bound into vm/buf cache
- Remove the syncer hooks in the buffer cache
- per-filesystem sync rather than per-vnode and per-filesystem
- Periodic sync only flushes metadata and updates inode times, let the file data flush lazily.

Fallout

- All of the io throttling and delaying needs to be reconsidered and re-tested
- Buffer cache contains only metadata
  - Must size accordingly
    - What about filesystems that don't use the new method?
  - Metadata no longer competing with data for buf space may be better
- More mapping overhead on 32bit systems
  - Negligent for page-by-page anyway