

Cache-Bypassing AFS Client

Matt Benjamin <matt@linuxbox.com>

May 21, 2008

New AFS Caching Concept: *Selective Caching*

- ✓ Selectively...
- ✓ Bypass OpenAFS Dcache Layer (UFS or Memcache)
 - ✓ Combine with UFS Cache for Fast Path with generally Persistent Cache
 - ✓ Combine with Memcache for... Faster Memcache

Motivation

- ✓ Avoid Disk I/O Penalty for Transient or Uncacheable Content (eg, Streaming)
- ✓ Avoid Caching Penalty for Cache Defeating Access Patterns
 - ✓ Primary Motivation in 2007
- ✓ Get Throughput Improvement

History

- ✓ MS Thesis: Thomas J. Hacker (Michigan, 1993)
- ✓ VAFS: NRL-Funded Research (CITI, 1998-2000)
 - ✓ Charles Antonelli, Kevin Coffman, Jim Rees, (Marcus Watts [Split])
 - ✓ <http://www.citi.umich.edu/projects/vafs/>
- ✓ Embian (www.embian.com)

Properties VAFS Cache Bypass

- ✓ Complete Rewrite
- ✓ Targeted Solaris 2.6 and Irix 5.x (SVR4.2)
- ✓ Aimed at General Throughput Improvements
 - ✓ Cache
 - ✓ Read Ahead
- ✓ Achieved ~20% Performance Improvement
 - ✓ Over Dcache?
 - ✓ Much of the Improvement Credited to Dcache Lock Contention Fixes
- ✓ Noted for odd odd criteria for bypass selection (server involvement)
 - ✓ but there is no universal choice (feedback welcome)

Embian

- ✓ Has built a significant web storage aggregation application using OpenAFS as a base
- ✓ More than .5 PB Storage On Line
- ✓ Clients are “Storage Middleware” Servers
 - ✓ Reliant on Fast Transfers
 - ✓ Cache Defeating Access Patterns
- ✓ Sungjin Chun, Dongguen Choi, Arum Yoon
 - ✓ OpenAFS/Kerberos Best Practices 2007
 - ✓ <http://workshop.openafs.org/afsbpw07/talks/chunsj.pdf>

2008 Cache Bypass

- ✓ Descendant of VAFS cache bypass
- ✓ Selective Caching Based on File-Size Threshold
- ✓ Improves Throughput of Already Well Tuned Memcache Clients
 - ✓ Heavy, Read-Intensive Workload
 - ✓ Cache Defeating Access Pattern (*Cache Penalty*)
- ✓ Targets Linux 2.6 (2.4 w/o Read Ahead)
 - ✓ remains intrinsically portable and Solaris port is reusable
- ✓ Weak Read Ahead algorithm in VAFS CBP removed in favor of Linux mechanism—for other platforms, generic adaptive mechanism called for

2008 Bypass Wins

- ✓ Cache Directly to Page Cache (Omit Multiple Caching, Copies w/Memcache)
- ✓ Uses Background Queue for Every I/O
 - ✓ More Parallel Fetch Opportunities
 - ✓ Better Interleaving of I/O Operations
- ✓ Better Read Ahead (On Linux, uses readpages VMOPs interface)
- ✓ Better SMP Scalability In General
 - ✓ KMAP is atomic, avoids reschedule and TLB flush
 - ✓ Omits many uses of BKL (Memcache Only)
 - ✓ Possibly better behavior in other ways we haven't discovered in profiling
- ✓ Adds code (separately submitted) to bypass RX_MAXCALLS limit (which we hit)

How Does It Work (Top Half)?

- ✓ Cache/No-cache strategy established on readpage or readpages VMOP
- ✓ If cache strategy changed, force sync-flush of Dcache before proceeding
- ✓ If bypassing, arguments checked, then I/O diverted to immediate background fetch
- ✓ Page-in call returns success immediately
 - ✓ Required pages in locked state

How Does It Work (Bottom Half)?

- ✓ When BOP_PREFETCH schedules, starts FetchData on file server
- ✓ Read length varies from 4-32 pages (ie, up to 256K on x86_64)
 - ✓ NoCacheFetchProc used to read data into VM page memory, unlocking pages as reads complete
 - ✓ Target pages mapped on-demand
 - ✓ current code strictly limits page mappings
 - ✓ mapped with kmap_atomic, substantial efficiencies gained on SMP systems

Additional Changes

- ✓ Usual flush to Dcache skipped if bypassing for a specific inode
- ✓ Write to mapped page triggers TransitionToCaching—pages may be flushed to file server as normal, on on subsequent TransitionToBypass
- ✓ Efficiency of latter process somewhat in doubt, since 2001—not yet demonstrated by profiling to be a problem, but probably worth replacing
 - ✓ Note that some write-behind mechanism desirable for most or all workloads

Results: Does Significantly Improve Read Throughput

- ✓ Results derived from client testing on a variety of 32-bit and 64-bit Intel platforms, with up to 4 CPUs
- ✓ On SMP hardware, bypass increased throughput 20%
- ✓ Adding Rx connection clones, performance increased an additional 20-25%
- ✓ Total speed-up from this effort: 44% +

Some Issues

- ✓ Cache Bypassed for Reads only
 - ✓ Odd transition mechanism created for CITI VAFS carried forward
 - ✓ Antonelli comment about equivalence to synchronous writes suggests
 - ✓ Concern about RX_MAXCALLS, certainly
 - ✓ Concerns about background queue capacity?
 - ✓ Concerns about Rx itself?

Planned Features

- ✓ Cache Bypass for Writes
 - ✓ Not equivalent to synchronous writes
 - ✓ Required to avoid spurious caching transitions
 - ✓ Potentially able to gain efficiencies comparable to Read CBP, although might be masked by longer I/O periods
 - ✓ Must integrate more with vcache
 - ✓ General efficiencies mentioned in OpenAFS Roadmap

Inclusion In OpenAFS

- ✓ Ideas Can Be Borrowed
 - ✓ Certainly will achieve this
- ✓ Acceptance Requires Closer Consideration of Internal Cache Manger APIs (Feedback)
 - ✓ Maintainability
 - ✓ Build Consensus

Cache Manager APIs: It Has Them?

- ✓ BG Queue
 - ✓ We are orthodox (increase its utilization)
- ✓ Vcache
 - ✓ We are orthodox
- ✓ Rx Interface
 - ✓ We are orthodox, albeit, operating outside the Dcache (new locking interactions possible)
- ✓ VFS/Dcache
 - ✓ Currently, (typically) a macro which resolves into a into UFS or Memcache
 - ✓ Options include: calling through intermediate layer, moving bypassing into more difficult Dcache functions, getting to like Bypass (perhaps with small changes in `afs_linux_readpage`, `afs_linux_readpages`)

Related CITI Work

- ✓ AFS Split-Path
 - ✓ Dedicated Bulk Data (FetchData, StoreData) Interface over ATM
 - ✓ In the Cache Manager and File Server
 - ✓ Large Performance Improvements Reported vs. 2001 Transarc CM
 - ✓ Worth Revisiting in 2008

Q & A