Finding the Needle Improving Management of Large Lustre File Systems



Presented by David Dillow





The Haystack

- Key metric for common management tasks is number of files rather than size
 - Generating candidates for sweep
 - Listing files on an OST
 - Naming inodes from error messages
- File systems keep growing
 - ORNL currently at 118M/62M/13M files
 - Down from peaks of 250+M/109M/86M files





Current Tools

- High-level, likely remote, access
 - find
 - Ifs find
- Low-level, direct access to LUNs
 - debugfs
 - e2scan
 - ne2scan





Execution environment

- Storage
 - Engenio 7900 (XBB2)
 - 80 1 TB SATA drives in RAID10
- Production
 - 4x 4Gb FC from host
 - 16 cores Xeon (2.4 GHz) w/ 64 GB memory
- Test bed
 - SRP over DDR Infiniband
 - 4 core Xeon (2.6 GHz) w/ 6 GB memory





Sweep takes longer and longer

- Up to nearly 21 hours for generating candidate list @ 150M files
 - Occasionally 8 hours
- 6+ hours for 90M
 - Occasionally 18 hours
 - Sometimes 4 hours
- Past usage affects future performance
 - As does current usage!





Finding files on an OST

- lfs find takes over 5 days at ~200M files
 - Live system
- fsfind takes 3h 20m to scan 150M files
 - Not including time to generate the 30GB list
 - Running over Lustre





Naming an inode

• ... Found existing inode

.../10979900/3759376635 state 7 in lock: setting data to

- .../10979916/3759376661
- debugfs takes just under 2 hours to name the inodes involved
 - This is from an idle system
 - Use -c to avoid reading bitmaps
 - Give it all inodes to name in one go





Why aren't the current tools faster?

- Client side
 - Round trip latency for operations
 - Difficulty in pipelining to hide latency
- libext2fs I/O behavior
 - Poor readahead behavior
 - Low queue depth
 - Small request sizes
 - Introduction of random seeks





Anatomy of (n)e2scan

- Phase 1 find interesting inodes
 - Linear scan of inode table
 - Iterate block list on directories
- Phase 2 name inodes
 - Scan directory locks and build a path tree
 - Simple case linear scan of directory content
 - Normal case highly random inode lookup





Feeding the Beast

- Modern IO systems thrive on
 - Larger requests
 - More requests in flight
- Kernel posix_fadvise() Qol issues
- Want to reuse improved IO patterns in other tools
- Don't want to re-write libext2fs





AIO IO Manager

- Uses native Linux AIO interface (libaio.a)
- **Reuses** io_channel_readahead()
- Uses O_DIRECT to improve control of requests
- Maintains internal cache
- Tunable cache size (default 20 MB)
- Tunable requests in flight (default 8 requests)





Initial Impact

- Inode Scan (150M inodes, no block iteration)
 - Unix IO manager 375 seconds
 - AIO IO manager 327 seconds
 - Both w/ block iteration 2445 seconds
- Directory Scan (80 GB data, name only)
 - Unix IO manager 3436 seconds
 - AIO IO manager 508 seconds





Addressing seeks

- Often don't need results immediately
- Already scheduling readahead requests
 - Easy to insert lazy requests into readahead stream





Basic Async API

- io_channel_async_read(channel, block, callback, priv1, priv2)
- io_channel_finish_async(channel, max_reqs)
- io_channel_async_count(channel)





High-level Async API

- ext2fs_block_iterate_async(fs, ino, inode, iter_callback, end_callback, priv)
- ext2fs_read_inode_async(fs, ino, async_state, callback, priv)





Improving the Async Behavior

- Original implementation rather naive
- Problems surfaced during path resolution phase
 - Read each inode block 8 times
 - Requested single block in each request
 - At least we kept the queue depth high...





Improving the Async Behavior

- Added duplicate request handling
 - Only send one request for a block
- Added request merging
 - Grow small requests into larger ones (~15% walltime)
 - Allow for a tunable gap between requests (~5% walltime)
- Improved cache and async bookkeeping





Impact of Async API

- Inode Scan (150M files w/ block iteration)
 - Unix IO manager 902 seconds (sync)
 - AIO IO manager 366 seconds
- Directory Scan (80GB w/ inode lookup)
 - Unix IO manager 51198 seconds (sync)
 - (n)e2scan 14709 seconds (sync)
 - AIO IO manager 3052 seconds





Improving the pathname resolution

- IO is now pretty efficient, but seeing stalls
 - We hit 100% CPU for several seconds at a time
 - Consequently, we stop tending to our IO requests





Improving the pathname resolution

- Replaced glibc rbtree with kernel's implementation
 - Worth ~15% CPU, 16.5% walltime
- Improved pathname generation
 - Avoid multiple walks of the path tree
 - Worth 30% CPU on micro-benchmark, ~5% on walltime





Current state

- Generates fslist compatible output
 - Full metadata dump
 - 160M files in 1 hour on testbed (vs 5 hrs)
 - 119M files in 35 to 50 minutes on production hardware (vs 3.5 to 8 hrs)
- Naming inodes
 - Picked three random inodes in a directory
 - Named in 15 minutes (vs 2 hrs)





Future work

- Clean up and release!
- Track down remaining CPU hogs
- Add listing of files on particular OSTs
- Add write support to AIO IO manager
- Proof-of-concept test for fsck





Questions?

 Contact info: David Dillow
865-241-6602
dillowda@ornl.gov



