Journaling Soft Updates

Brought to you by

Dr. Marshall Kirk McKusick
and
Jeffery Roberson

BSD Canada Conference 2010
14th May 2010

Overview

- Introduction to soft updates
- Filesystem operations that require journaling
- Additional requirements of journaling
- Crash recovery
- Performance
- Project status

Keeping Metadata Consistent 1

- Synchronous writes
  - Benefits: simple and effective
  - Drawbacks: create/delete intensive applications run slowly, slow recovery after a crash
- Non-Volatile RAM
  - Benefits: usually runs all operations at memory speed, quick recovery after a crash
  - Drawbacks: expensive hardware unavailable on many machines, somewhat complex recovery
- Atomic Updates (journaling and logging)
  - Benefits: create/remove do not slow down under under heavy load, quick recovery after a crash
  - Drawbacks: extra I/O generated, little speed-up for light loads

Keeping Metadata Consistent 2

- Copy-on-write Filesystem (LFS, ZFS, WAFL, etc)
  - Benefits: write throughput, cheap snapshots, always consistent
  - Drawbacks: disk fragmentation, memory overhead
- Soft updates
  - Benefits: most operations run at memory speed, reduced system I/O, instant recovery after a crash
  - Drawbacks: complex code, background fsck, and increased memory loading

Tracking File Removal Dependencies

Ordering constraints

1) Name in on-disk directory must be deleted
2) Deallocate (zero out) on-disk inode
3) Release file’s blocks to free-space bitmap

How soft updates maintains this ordering

1) Zero out directory entry in kernel buffer and hang a dependency structure on buffer to be notified when buffer is written.
2) When notified that directory buffer is written, save list of inode’s blocks, then zero out inode in kernel buffer and hang a dependency structure (containing the list of blocks) on buffer to be notified when buffer is written.
3) When notified that inode buffer is written, release list of saved blocks to free-space bitmap.

Recovery After a Crash

- Disk state is always valid but behind in-memory state
- Only inconsistencies:
  - Blocks marked in use that are free
  - Inodes marked in use that are free
- It is safe to run immediately after a crash though eventually lost space must be reclaimed

Adding Journaling to Soft Updates

Only need to journal operations that orphan resources

Journal needs a maximum of 16Mb independent of filesystem size

Filesystem operations that require journaling

- Increased link count
- Decreased link count
- Unlink while referenced
- Change of directory offset
- Cylinder group updates of freed blocks and inodes

Additional Requirements of Journaling

Additional soft update tracking

- Cylinder group rollbacks
- Additional inode rollbacks

Reclaiming journal space

- Soft-update dependencies reference oldest segment-structure in the journal with entries that describe the operation
- Release journal segment when all dependency references to it are gone

Unlinked inode list

Crash recovery

Crash recovery is done by *fsck*

- Recovery is idempotent
- No filesystem updates made until changes are fully verified

Recovery steps

- Scan the journal
- Link count increases
- Link count decreases
- Free inodes with zero link count
- Free inodes that were unlinked but busy
- Free unallocated blocks

Performance

Recovery times

- Eight-way buildworld on 250GB 80%-full disk reset after 10 minutes
  - Journal recovery: 0.9 seconds
  - Verification *fsck*: 27 minutes
- Random collection of parallel file writes on 11Tb 92%-full 14-disk 3ware RAID array reset after several hundred megabytes of written data
  - Journal recovery: under a minute
  - Verification *fsck*: 10 hours

Runtime slowdown

- Additional I/O to journal
- Blocking to wait for journal writes
- Extra CPU overhead is negligible

Project Status

- Merged into head of tree

  Known bugs have been fixed

  Performance work in progress

- Ports to 6-stable, 7-stable, and 8-stable in projects/suj/{6,7,8} though will not be MFC’ed to any of these trees

- Enabled using tunefs, requires that 16Mb of space is available to create the journal

- Old kernels see a dirty filesystem if journaling was in use and a rollback has not been done

- Old kernels clear journaling flag so new journaling kernel knows to clear journal and check a dirty filesystem before running on it

Interesting Statistics

- Eleven new dependency types were added to the existing fourteen
- Nearly doubled size of soft-updates code (6,409 lines to 11,491 lines)
- Journal recovery code is 2,600 lines versus 6,100 for background *fsck*
- Journal recovery code shares little of the normal *fsck* code
- Adding the new dependencies was easy compared to figuring out and adding the new rollbacks
- Directory rename consumed a quarter of the development time

Questions

Journaled Soft-updates paper:

http://www.mckusick.com/publications/suj.pdf

Journaled Soft-updates slides:

http://www.mckusick.com/publications/suj-slides.pdf

Marshall Kirk McKusick

<mckusick@mckusick.com>

Jeffery Roberson

<jroberson@jroberson.net>