# Devsummit – Notes on WAPBL

Taylor ‘Riastradh’ Campbell
campbell@mumble.net
riastradh@NetBSD.org

EuroBSDcon 2015
Stockholm, Sweden
October 2, 2015

# WAPBL — Write-Ahead Physical Block Logging

- Write-ahead physical block logging.
- Reduces metadata write latency.
- Reduces time to mount after crash.
- Generic framework, used only by `ffs` at the moment.

# Traditional `ffs`

- Synchronous metadata block writes:
  - Find data blocks in freelist, mark as allocated.
  - Set inode's data block pointers.
- Every step keeps the file system state *consistent* but not *clean*.
- If crash happens in middle, fsck globally analyzes file system to find allocated-but-unreferenced data blocks, etc.
- Problem: synchronous metadata has high latency.
- Problem: fsck must globally analyze file system—slow to pick up again after crash.

# Logging

- Asynchronous metadata block writes, but serialized via write-ahead log.
- Write metadata blocks to write-ahead log first.
- Flush log blocks to disk.
- Write flushed log blocks to real location in disk.
- Mark log blocks committed.
- If crash happens in middle, replay uncommitted log blocks.

# Not all physical block logging

- Mostly log has just physical blocks: verbatim copy of block to write elsewhere.
- Some operations too complex to handle this way.
- Inode allocation: log a record marking inode number as pending allocation; then do complex inode allocation logic; then log a record marking it as allocated.
- If crash in middle: undo all pending-allocation inode records on mount.
- Block deallocation: can't reallocate blocks until log flush happens.

# Problem: tentacles

- Needed tentacles inside `buf(9)` abstraction, `vfs_bio.c`.
- Needed tentacles inside UVM unified buffer cache getpages/putpages, `genfs_io.c`.
- Every `ufs_write` happens inside a single transaction. (Data blocks not logged—but wapbl transaction lock held across all data writes via putpages anyway.)

# Problem: truncation

- Log is bounded size.
- Log transactions are bounded size.
- Truncate large file: need to deallocate each block *and* truncate inode.
- So `ufs_truncate` truncates one indirect block at a time.
- But a 1 TB file has a lot of indirect blocks—and truncation is one block at a time even if file is sparse!
- Patch floating around to do as much in a single transaction as possible.
- Better algorithm: truncate only allocated blocks. (But requires some bookkeeping to get right.)

# Problem: appending data

- To append to a file:
  - Allocate data blocks (logged metadata write).
  - Increase inode size (logged metadata write).
  - Write data blocks (asynchronous data writes).
- No ordering between metadata and data blocks.
- If crash after metadata writes before data writes, file may appear to have garbage data from free blocks appended!