Alamosa: A Tiered Disk Cache for NetBSD

Problem Statement

- Slow disk is large, fast disk is small
- Workloads not always easy to partition into pieces
  - You don’t know in advance which records will be hot
- Linux has bcache
- ZFS has some similar capabilities that don’t quite scratch the itch (ZIL, L2ARC)
  - And also, I felt like writing a block device driver

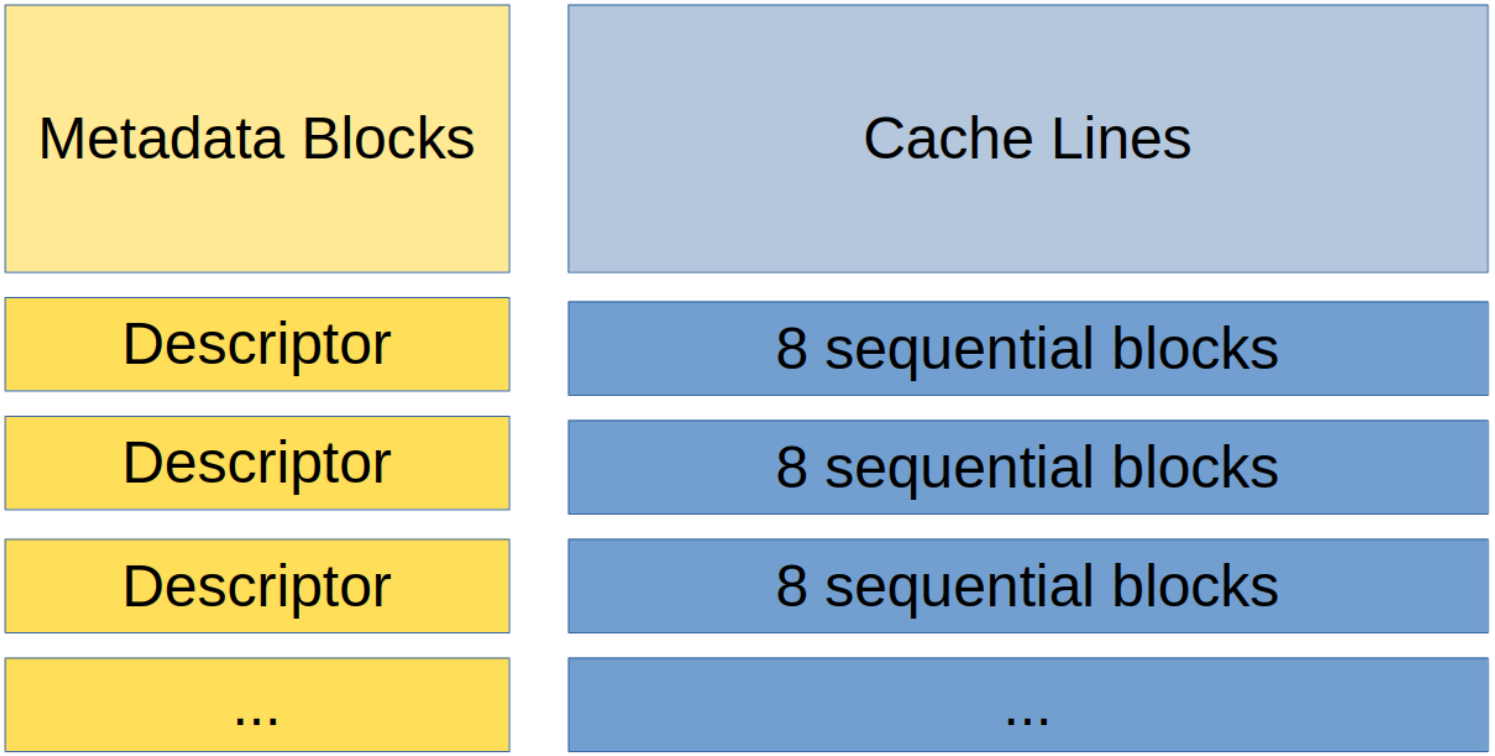
Target Workloads

- Ultimately, will be used on production server hosting metadata for a embedded device
  - sqlite (some)
  - lmdb (lots)
- Workload characteristics
  - Almost entirely reads
  - Write performance was not a priority

Early Prototypes

- I experimented a lot with fancy designs at first
  - Large LRU structures
  - Elaborate free-space management structures
    - … then realized I don’t need these
- Eventual realization: it’s a cache, it can work like one
  - Free space management? Nah
  - Large-scale LRU? No way
  - All that matters is tracking clean and dirty

The Original Alamosa Design



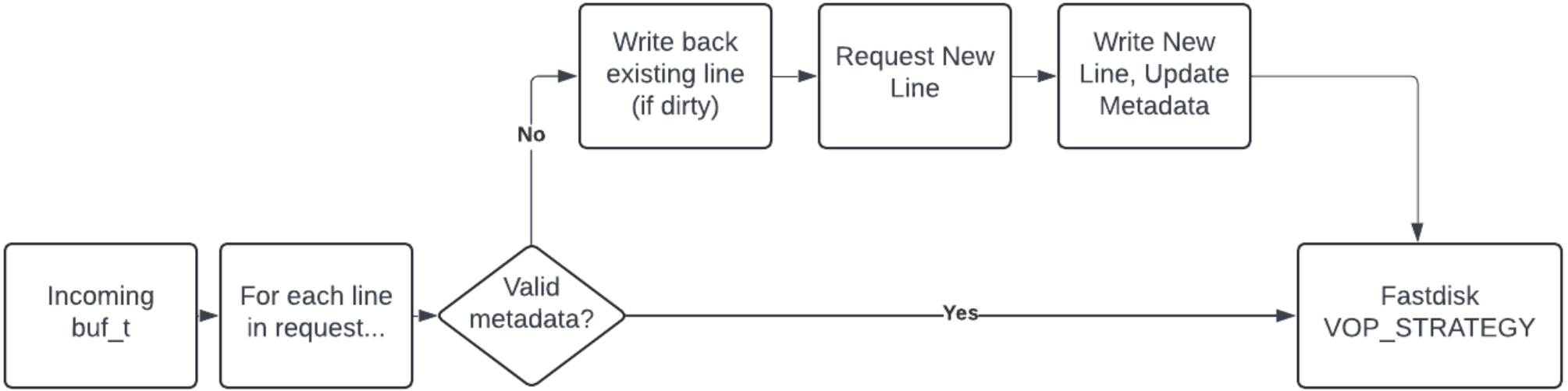
Metadata

- A metadata block has metadata lines
  - Each one describes a full cache line – eight blocks
  - These go to/from the slow disk as one
- A metadata line is 64 bits – 0:59 is line base addr, 60-61 is reserved, 62 is valid, 63 is dirty
- Metadata occupies the beginning portion of the fast disk and is contiguous

Cache Lines and Lookup

- A cache line's base address is hashed, producing a index into both the cache-line region and the metadata region
- Why store lines?
  - Many workloads are at least a little bit sequential
  - Shoot for the best ratio of metadata to cache data – allows more effective storage capacity!
  - One 64b descriptor describing eight disk blocks is a good tradeoff

Block Lookup Flow



Implementation Adventures

- NetBSD has had a few generations of disk and block I/O interfaces
- It also has kernel autoconfiguration, for clean management of devices and their relationships
- For the most part, I did not use these interfaces
  - The ccd driver largely uses older interfaces, and I started off as a ccd derivative because it seemed spiritually similar to what I'm doing. :)
  - In retrospect, cgd might have been a better basis…
  - Right now, configuration entirely ioctl driven

Implementation Adventures, Part 2

- A cycle: Build, crash kernel, get stuck in angry FFS fsck, repeat
- I should have used rump kernels for this
  - … but I didn’t
  - Next time!

Design Limitations

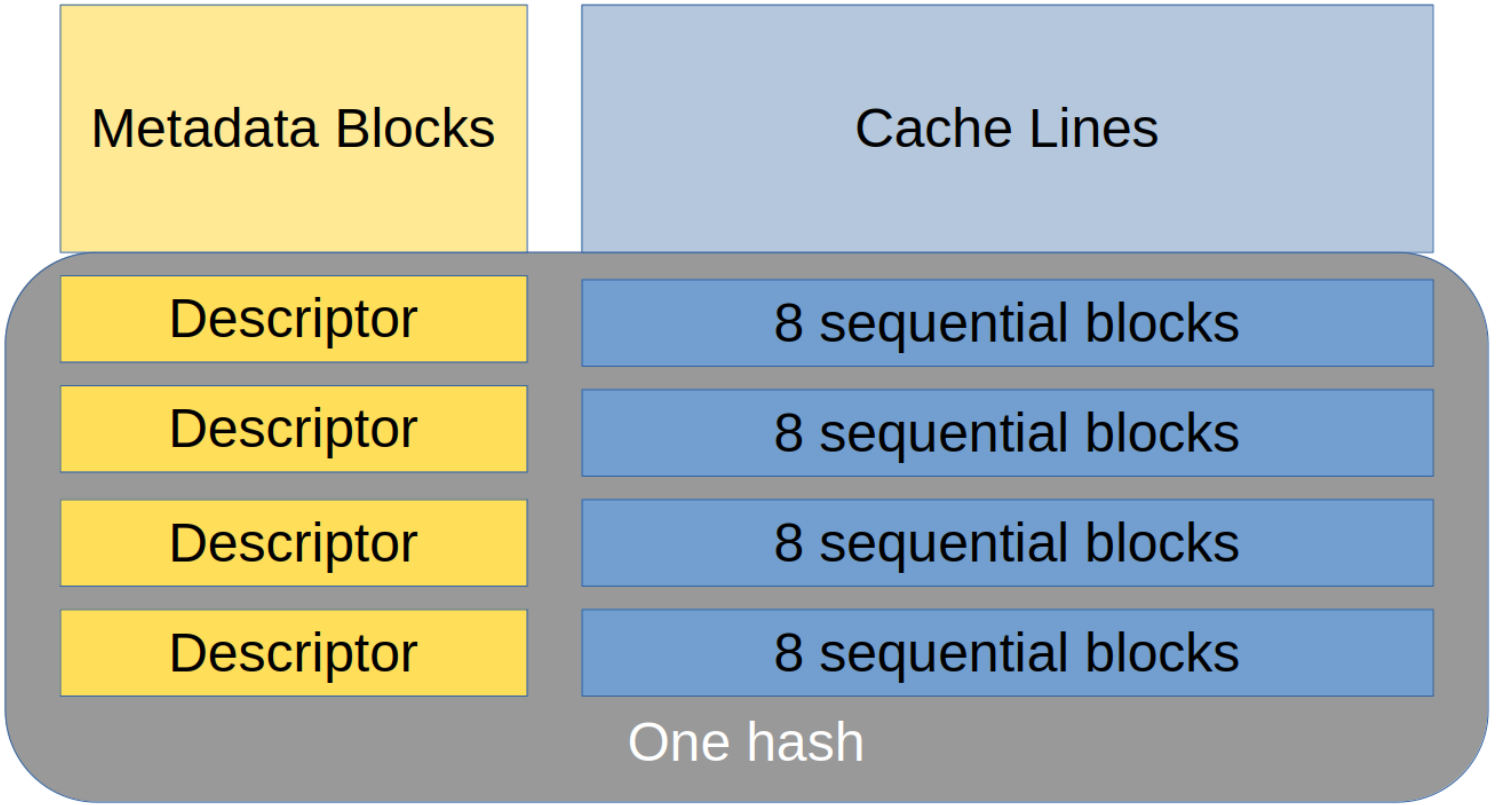
- Contention from direct-mapped design kills performance on target workloads
- Not written to be tunable
- “Baby’s first block driver” – implementation is not great
  - No autoconf
  - Writeback should be handled via queues
- Write performance was never a priority

Back to the drawing board...

Alamosa2 to the Rescue!

- Direct-mapped LRU replaced with new multi-way structure
  - 4-way associativity default – that is, four lines of same hash can simultaneously exist in the cache
  - Intended to be configurable
- Eviction policy can be random
  - … or modified random – track x recently-used lines in an in-memory structure, choose randomly between older lines
- Persistent LRU is probably not desirable

Alamosa2 Structure



New Driver Features

- Proper kernel autoconf!
- LWP-based writeback and flexible locking
- Configurability
  - Eventual goal – “generate a tuned profile by running test workloads for a few minutes”
  - Higher associativity potentially gives higher hit rate, but has tradeoffs
  - Line size – longer lines can mean higher hit rate or lower depending on access locality

Lessons Learned for Alamosa2

- Use rump kernels when you can
  - Use scripted VMs from a standard image when you can’t
- Test early
- Test often
- Automate
- If building and testing module is a Major Process, you’re doing it wrong

The Road to Upstream

- Get Alamosa2 stable!
  - … make it run production workloads on two platforms for at least a couple of months without eating data!
  - Throw strange stuff at it! Try to break it!
- Do performance characterization!
- Freeze the design!

The Road to Upstream: Part 2

- Once everything is working, have the upstream conversation
- Early 2025? Here's hoping!
- Alamosa is being developed alongside some other components
  - Dual-kernel realtime
    - Like Xenomai
  - ia64 fixes
    - Hoping to boot on real hardware (rx2800)
    - NVMM? Maybe… but the QEMU dependency is rough
- Ideally, I'd like to upstream all of them eventually

Questions?