CS 333 :: Meeting Notes :: The B^{ε}-tree File Systems (BetrFS)

From B^{ε}-trees to BetrFS

B^{ε}-trees, like LSM-trees are an example of a write-optimized key-value store. BetrFS is a Linux file system that uses B^{ε}-trees as its on-disk data structure, radically departing from traditional inode-based designs. The BetrFS design has seen several iterations and is still under active development. We will focus on the first two versions, and discuss the trajectory of the ideas.

BetrFS v1

The key design principles of BetrFS are to:

- Avoid "*cryptosearches*". A cryptosearch is the term that the market researchers came up with in order to describe an *implicit search* that is triggered by an update. For example, if we try to write 15 bytes to the middle of a file, a traditional file system would first have to read the block(s) that contains those bytes, update the appropriate bytes in memory, and then write the new version of those blocks back to the disk. The cryptosearch is the fact that our "write" was turned into a "read-modify-write" despite the fact that we never explicitly asked to read the data.
  - To avoid cryptosearches, BetrFS tries to use "blind writes" whenever possible. Upserts enable blind writes in may situations: we may be able encode the final value without needing to know the initial state (e.g., overwrite 15 bytes starting at offset X, append to end of file), or we may be able to express the change as a function that takes some initial state as an input (e.g., update a counter in an inode)
- Design a schema that lets BetrFS efficiently map VFS operations to efficient B^{ε}-tree operations. In other words, we want each VFS operation to be mapped to the fewest number of B^{ε}-tree operations possible, and we want whatever B^{ε}-tree operations we do use to be efficient operations.
  - The BetrFS schema uses the full-paths of files to define its keys, and either the metadata (stat/inode) or a 4KiB data block as the value. There are two separate B^{ε}-trees: one for data and one for metadata.
    - In the data B^{ε}-tree, the keys are full paths, a "separator", and a block number. For example, `/home/bill/foo.txt:1` maps to block 1 of the file `/home/bill/foo.txt`
      - This design means that all data blocks for a file will be contiguous in the keyspace (and therefore contiguous in the leaf nodes)
    - In the metadata B^{ε}-tree, the key is just the full path. For example, `/home/bill/foo.txt`
    - This means that all metadata will be sorted in a breadth-first search order with respect to the file system directory structure
    - This also means that naively renaming or deleting an object will scale with the size of that object:
      1. Each data block must be moved from its old location in the tree structure to the new location, and there is no correspondence between B^{ε}-tree nodes and the tree structure that would let you just adjust pointers
      2. Each key must be modified to reflect the new path
      3. Each old value must be deleted

BetrFS v1 used B^{ε}-trees as a black box. It wanted to see how far the design could be pushed before changing (1) the OS and (2) the data structure.

BetrFS v2

BetrFS v2 set out to solve some of the biggest performance drawbacks in BetrFS v1. It introduced 3 new techniques:

- Range messages: range messages can be applied to multiple keys.
  - Range messages were used to speed up deletes by issuing one delete message regardless of the object size
- A late-binding journal. Logging data is key to B^{ε}-tree performance: small updates can be made persistent fast and enable batching in the data structure. But for large sequential writes, it may be cheaper to immediately write back a node rather than write the data twice (once to the log, and again in the tree).
- Tunable indirection with zones. Zones group regions of the file system directory tree, and the B^{ε}-trees keys are made relative to a zone's root
  - Renaming the root of the zone is fast, since no in-zone keys must be changed
  - Renaming within a zone is bounded by the size of the zone

BetrFS v2 still left the OS alone, but it made small changes to the data structure and schema to alleviate some of the BetrFS v1 bottlenecks. Unfortunately it compromised on the full-path keys, which future versions addressed with more invasive data structure questions.