

Fast File System

Don Porter





How to place a file system on disk?

- Let's assume we have the following:
 - Super block (allocation bitmap, FS-level metadata)
 - Inodes (file-level metadata)
 - Data blocks

• Thoughts?



Strawman

Super- block	Inodes	Data
0		DISKSZ

• Problems?



Typical file access pattern



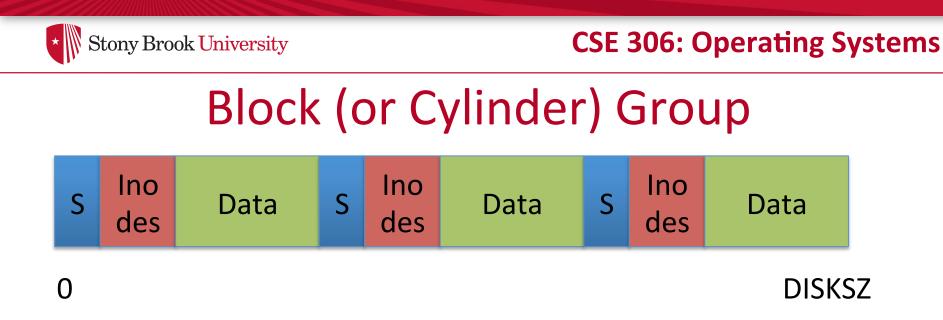
- cat a
- cat b
- cat c

Lots of seeking – no locality for head across files⁴



Metadata locality

- File data and metadata (inode) are frequently accessed together
- Simple design fails to capture this pattern
- Any ideas?

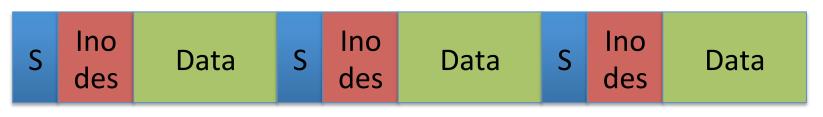


- Stripe smaller chunks of these triples across disk
- Superblock:
 - Some data replicated (good for crash tolerance)
 - Some data distributed (free block bitmap)
- Per-group inodes and blocks



 $\mathbf{0}$

Block (or Cylinder) Group



DISKSZ

- What does this give you?
 - Average case: Inode + data relatively close
 - Reduce average-case seek time
- Performance goal:
 - Put things together that are accessed together
 - How?





FFS data placement heuristics

- Keep related things together
- Keep unrelated things far apart
- Directories:
 - New directories placed in least-utilized cylinder group
 - Low number of total directories + plenty of free inodes
 - Why?
- Files:
 - Blocks of files should be allocated in same group as inode
 - Place files in same directory in same group

Edge cases?





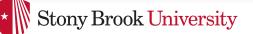
Edge case 1: Large files

- Where to place a big file (e.g., movie download)?
 - Option 1: Fill up 1+ entire block groups (best fit)
 - Pro: Data is close together
 - Con: Wastes inodes in block group (or causes them to be far apart
 - Requires a lot of free space in 1+ groups to work
 - Degenerate case: only a few blocks free in each group
 - Option 2: Spread data across many block groups (worst fit)
 - Pro: Tries to keep larger regions of free space
 - Cons: Can end up seeking across many block groups



Amortizing seeks

- Seeks have a fixed cost
 - Let's say 10 ms on a current HDD
- Transfer time is proportional to amount of contiguous data moved
 - Let's say 125 MB/s on a current HDD
- Insight: We can control fraction of time spent seeking by data allocation size



Amortizing seeks

- Suppose we want to spend half of our time seeking:
 - I.e., we want to spend 10 ms in transfer time

$$\frac{125MB}{1s} * \frac{1s}{1000ms} * 10ms = 1.25MB$$

Suppose we want to spend 10% of our time seeking

$$\frac{125MB}{1s} * \frac{1s}{1000ms} * 90ms = 11.25MB$$

Caveat: You need to actually use this much data¹





Fragmentation

- Not fragmenting free space becomes very important to performance
- Internal: Lots of files smaller than 1.25 MB
 - Idea: pack multiple small files into one 1.25 MB chunk
 - Called sub-blocking
- External: Need to keep enough free space in a block group for a directory
 - Approach: load balance across block groups
 - No good solution when disk is nearly full





Edge case 2: Renaming

- How does rename work?
 - Change the pointer from name to inode
- Implication for locality if you move files across directories?
 - Create in one block group
 - Rename to directory in a different block group
 - Directory contents no longer in same group





Edge case 2: Renaming

- What to do?
 - Live with it (one of several was a file system ages)
 - Move the data (slow):
 - BetrFS v.1 does this; takes 5 minutes to rename a 4 GB file



FFS Summary

- First file system designed for good performance
- Design principles still in use today
 - Ext* family on Linux
 - FFS still used in BSD
- Key ideas:
 - Block/cylinder groups
 - Data placement heuristics
 - Amortizing seeks