PERSISTENCE: FSCK, JOURNALING

Shivaram Venkataraman
CS 537, Spring 2019
Project 4b: Due today!

Project 5: Out by tomorrow

Discussion this week: Project 5
AGENDA / LEARNING OUTCOMES

How does FFS improve performance?

How to maintain consistency with power failures / crashes?
RECAP
FS STRUCTS: SUPERBLOCK

Basic FS configuration metadata, like block size, # of inodes

<table>
<thead>
<tr>
<th>S</th>
<th>IB</th>
<th>DB</th>
<th>I</th>
<th>I</th>
<th>I</th>
<th>I</th>
<th>I</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td>39</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td>55</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| D | D | D | D | D | D | D | D | D | D |
| 8 |    |    | 15 |
| D | D | D | D | D | D | D | D | D | D |
| 24 |    |    | 31 |
| D | D | D | D | D | D | D | D | D | D |
| 40 |    |    | 47 |
| D | D | D | D | D | D | D | D | D | D |
| 56 |    |    | 63 |
**Inode**

- type (file or dir?)
- uid (owner)
- rwx (permissions)
- size (in bytes)
- Blocks
- time (access)
- ctime (create)
- links_count (# paths)
- addrs[N] (N data blocks)
inode

**data**

**data**

**data**

**indirect**

*Imbalanced tree*

*Small file: direct pointer*

*Large file: indirect pointer*
Key idea: Keep inode close to data

Use groups across disks;

Strategy: allocate inodes and data blocks in same group.
POLICY SUMMARY

File inodes: allocate in *same* group with dir

Dir inodes: allocate in *new* group with fewer used inodes than average group

First data block: allocate near inode

Other data blocks: allocate near previous block
SPLITTING LARGE FILES

Define “large” as requiring an indirect block
Starting at indirect (e.g., after 48 KB) put blocks in a new block group.

Each chunk corresponds to one indirect block
Block size 4KB, 4 byte per address => 1024 address per indirect
1024*4KB = 4MB contiguous “chunk”
Assume that the average positioning time (i.e., seek and rotation) = 10 ms. Assume that disk transfers data at 100 MB/s.

If FFS large file chunk size is 4MB, what is the effective throughput we are getting?

\[
\text{read } 4\text{MB} = 40\text{ms} \\
\text{seek } = 10\text{ms}
\]

\[
\frac{40}{100} \times 100 = 80\text{ MB/s}
\]

What is the effective throughput with 8MB chunk size?

\[
\text{read } 8\text{MB} = 80\text{ms} \\
\text{seek } = 10\text{ms}
\]

\[
\frac{8}{9} \times 100 = 89\text{ MB/s}
\]
POLICY SUMMARY

File inodes: allocate in same group with dir
Dir inodes: allocate in new group with fewer used inodes than average group

First data block: allocate near inode
Other data blocks: allocate near previous block

Large file data blocks: after 48KB, go to new group.
Move to another group (w/ fewer than avg blocks) every subsequent 1MB.
OTHER FFS FEATURES

FFS also introduced several new features:

- large blocks (with libc buffering / fragments)
- long file names
- atomic rename
- symbolic links

hard links - Introduced by FFS
First disk-aware file system

- Bitmaps
- Locality groups
- Rotated superblocks
- Smart allocation policy

Inspired modern files systems, including ext2 and ext3
FILE SYSTEM CONSISTENCY
**FILE SYSTEM CONSISTENCY EXAMPLE**

**Superblock**: field contains total number of blocks in FS
DATA = N

**Inode**: field contains pointer to data block; possible DATA?
DATA in \{0, 1, 2, ..., N - 1\}

Pointers to block N or after are invalid!
Total-blocks field has redundancy with inode pointers
WHY IS CONSISTENCY CHALLENGING?

File system may perform several disk writes to redundant blocks

If file system is interrupted between writes, may leave data in inconsistent state

What can interrupt write operations?
- power loss
- kernel panic
- reboot
FILE APPEND EXAMPLE

Starting

append Db to this file

Just Db is written
≤ FS consistent

Just B is written
≤ space leak

Just I is written
≤ Garbage data bitmap
Inconsistency & Inode

Allocate new data block
update inode

Write data fields

Bitmap & Db are written
≤ data block is used?
Unreachable!
HOW CAN FILE SYSTEM FIX INCONSISTENCIES?

Solution #1:

FSCK = file system checker

Strategy:

After crash, scan whole disk for contradictions and “fix” if needed

Keep file system off-line until FSCK completes

For example, how to tell if data bitmap block is consistent?

Read every valid inode+indirect block

If pointer to data block, the corresponding bit should be 1; else bit is 0
FSCK CHECKS

Do superblocks match?
Is the list of free blocks correct?
Do number of dir entries equal inode link counts?
Do different inodes ever point to same block?
Are there any bad block pointers?
Do directories contain “.” and “..”?
...

Replicated Superblocks = make sure all super blocks are same
LINK COUNT EXAMPLE

Dir Entry ➔ inode

 inode
 link_count = 2

Dir Entry ➔ inode

hard links point to same inode
FREE BLOCKS EXAMPLE

inode
link_count = 1

data bitmap
0011001100

for block 123

block (number 123)

Inconsistent bitmap

Always trust the inodes and fix the bitmap

Consistent
Correct

Valid?
Duplicate Pointers

inode
link_count = 1

block
(number 123)

inode
link_count = 1

block
(number 769)

"Consistent"
BAD POINTER

inode
link_count = 1

super block
tot-blocks=8000

delete pointer?

9999 > tot_blocks
BUNNY 17

https://tinyurl.com/cs537-sp19-bunny17
FILE SYSTEM STATE: Consistent or inconsistent? If inconsistent, how to fix?

Inode Bitmap: 11111111
All are allocated?

Inode Table: [size=1, ptr=0, type=d] [] [] [] [] [] [] []

Data Bitmap: 10000000

Data: ["." 0), (".." 0)] [] [] [] [] [] [] []

Inconsistent. Clear the bits in Inode bitmap that are 10000000

Inode Bitmap: 11000000

Inode Table: [size=1, ptr=0, type=d] [size=1, ptr=1, type=d] [] [] [] [] [] [] []

Data Bitmap: 11000000

Data: ["." 0), (".." 0), ("a" 1)] ["." 1), (".." 1)] [] [] [] [] [] [] []

Bitmaps are consistent. Dir entry in block 1
PROBLEMS WITH FSCK

Problem 1:
- Not always obvious how to fix file system image
- Don’t know “correct” state, just consistent one
- Easy way to get consistency: reformat disk!
PROBLEM 2: FSCK IS VERY SLOW

Checking a 600GB disk takes \(~70\) minutes

ffsck: The Fast File System Checker
Ao Ma, Chris Dragga, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau
CONSISTENCY SOLUTION #2: JOURNALING

Goals

– Ok to do some **recovery work** after crash, but not to read entire disk
– Don’t move file system to just any consistent state, get **correct** state

Atomicity

– Definition of atomicity for **concurrency**: operations in critical sections are not interrupted by operations on related critical sections
– Definition of atomicity for **persistence**: collections of writes are not interrupted by crashes; either (all new) or (all old) data is visible
CONSISTENCY VS ATOMICITY

Say a set of writes moves the disk from state A to B

fsck gives consistency
Atomicity gives A or B.
JOURNAL LAYOUT

Super | Journal | Group 0 | Group 1 | ... | Group N

Transaction

Inode Bmap | Data Bmap | Inodes | Data Blocks

Journal | TxB | I[v2] | B[v2] | Db | TxE

Transaction
transaction: write A to block 5; write B to block 2

Checkpoint: Writing new data to in-place locations
JOURNAL REUSE AND CHECKPOINTS

transaction: write A to block 5; write B to block 2

Checkpoint: Writing new data to in-place locations

transaction: write C to block 4; write T to block 6
Ordering for Consistency

Transaction: write C to block 4; write T to block 6
Write order: 9, 10, 11, 12, 4, 6

Barriers
1) Before journal commit, ensure journal transaction entries complete
2) Before checkpoint, ensure journal commit complete
3) Before free journal, ensure in-place updates complete
CHECKSUM OPTIMIZATION

Can we get rid of barrier between (9, 10, 11) and 12?

write order: 9, 10, 11 | 12 | 4, 6 | 12

In last transaction block, store checksum of rest of transaction
During recovery: If checksum does not match, treat as not valid
OTHER OPTIMIZATIONS

Batched updates
- If two files are created, inode bitmap, inode etc. get written twice
- Mark as dirty in-memory and batch updates

Circular log

Journal: 0 128 MB
HOW TO AVOID WRITING ALL DISK BLOCKS TWICE?

Observation: Most of writes are user data (esp sequential writes)

Strategy: journal all metadata, including
superblock, bitmaps, inodes, indrects, directories

For regular data, write it back whenever convenient.
transaction: append to inode I
Crash !?!
ORDERED JOURNALING

Still only journal metadata

But write data **before** the transaction!
What happens if crash now?
B indicates D currently free, I does not point to D;
Lose D, but that might be acceptable
SUMMARY

Crash consistency: Important problem in filesystem design!

Two main approaches

FSCK:
   Fix file system image after crash happens
   Too slow and only ensures consistency

Journaling
   Write a transaction before in-place updates
   Checksum, batching, ordered journal optimizations
Next class: How to create a file system optimized for writes

Project 4b due today!
Discussion on Thu: Project 5