PERSISTENCE: FSCK, JOURNALING

Shivaram Venkataraman
CS 537, Spring 2023
Project 6 updates

Midterm 2: Solutions, grades

No class on Tuesday!
AGENDA / LEARNING OUTCOMES

How to check for consistency with power failures / crashes?

How to ensure consistency in filesystem design?
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>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>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>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>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</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>24</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>40</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>56</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>
</tbody>
</table>

Basic FS configuration metadata, like block size, # of inodes
<table>
<thead>
<tr>
<th>TIME</th>
<th>data bitmap</th>
<th>inode bitmap</th>
<th>root inode</th>
<th>foo inode</th>
<th>bar inode</th>
<th>root data</th>
<th>foo data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1. read</td>
<td></td>
<td></td>
<td>2. read</td>
<td>4. read</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3. read</td>
<td></td>
<td>5. read</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7. write</td>
<td></td>
<td>8. read</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6. write</td>
<td>9. write</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10. write</td>
<td></td>
<td>11. write</td>
</tr>
</tbody>
</table>

create /foo/bar
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

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

Inspired modern file 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

Inode Bmap | Data Bmap | Inodes | Data Blocks
--- | --- | --- | ---

--- | --- | I[v1] | Da

--- | --- | I[v2] | Da Db
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 “..”? 
...

FREE BLOCKS EXAMPLE

inode
link_count = 1

block
(number 123)

data bitmap
0011001100

for block 123
LINK COUNT EXAMPLE

- Dir Entry
- Dir Entry
-_inode
  - link_count = 1
DUPLICATE POINTERS

inode
link_count = 1

block
(number 123)

inode
link_count = 1
BAD POINTER

inode
link_count = 1

super block
tot-blocks=8000

9999
(a) FILE SYSTEM STATE: Consistent or inconsistent? If inconsistent, how to fix?

Inode Bitmap : 11111111
Inode Table : [size=1,ptr=0,type=d] [] [] [] [] [] [] [] []
Data Bitmap : 10000000
Data : ["." 0), (".." 0)] [] [] [] [] [] [] []
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), ["." 0)] [] [] [] [] [] []

Inode Bitmap : 11100000
Inode Table : [size=1,ptr=0,type=d] [size=1,ptr=1,type=r] [size=1,ptr=2,type=r] [] [] [] [] [] []
Data Bitmap : 11100000
Data : [ ["." 0), ["." 0)] [DATA] [DATA] [] [] [] [] [] []
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

Inode Bmap | Data Bmap | Inodes | Data Blocks

Transaction

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

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

write order
9, 10, 11

12

4, 6
Can we get rid of barrier between (9, 10, 11) and 12?

In last transaction block, store checksum of rest of transaction.

During recovery: If checksum does not match, treat as not valid.
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: T1  T2  T3  T4
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, indirects, directories

For regular data, write it back whenever convenient.
transaction: append to inode I

Crash !?!
Still only journal metadata. But write data **before** the transaction!

What happens if crash in between?

write order

7
9, 10, 11
12
2, 4
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 STEPS

No class on Tuesday!

Next time we meet: How to create a file system optimized for writes