PERSISTENCE: FSCK, JOURNALING

Shivaram Venkataraman
CS 537, Spring 2020
ADMINISTRIVIA

Project 5: Are you working on it?!  

Discussion this week: Practice for the final
AGENDA / LEARNING OUTCOMES

How to check for consistency with power failures / crashes?

How to ensure consistency in filesystem design?
**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>Inode</th>
<th>Data Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>39</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>55</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Block sizes: 8, 15, 23, 31, 39, 47, 55, 63
create /foo/bar

1. read
2. read
3. read
4. read
5. read
6. write
7. write
8. read
9. write
10. write

10 steps to create a file: inode, data, etc.

Parent directory for bar

last modified
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.
FILE SYSTEM CONSISTENCY
**FILE SYSTEM CONSISTENCY EXAMPLE**

**Superblock**: field contains total number of blocks in FS  
DATA = N \sim 1024 \text{ in } x \times 6

**Inode**: field contains pointer to data block; possible DATA?  
DATA in \{0, 1, 2, \ldots, 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 → OS crashed
- reboot
FILE APPEND EXAMPLE

Write 3 blocks to disk

1 date block

① Only data block D₃ is on disk → Can't reach the data

② Only inode is updated on disk → Pointing to garbage data → Bitmap inconsistent

③ Only bitmap is updated → Bitmap inconsistent But no inode points to it

Inode | Data  | Inodes | Data Blocks
---|---|---|---
Bmap | Bmap | l[v1] | Da

3 changes

Inode | Data | Inodes | Data Blocks
---|---|---|---
Bmap | Bmap | l[v2] | Da Db

Used
olution #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 “..”?

…
One way to fix this is to set the bitmap to 1.
LINK COUNT EXAMPLE

inode
link_count = 1

But 2 dir entries point to this inode

But 2 dir entries point to this inode

(link count)

But 2 dir entries point to this inode

(common fix is to make the count = 2)
**DUPLICATE POINTERS**

- **inode**
  - link_count = 1

- **block**
  - (number 123)

- **inode**
  - link_count = 1

---

Not correct? But consistent

Only one inode should point to a block

- **block**
  - (number 769)
BAD POINTER

inode
link_count = 1

super block
tot-blocks=8000

Can't access block 9999

Common fix: remove the addr which is 78000
(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]] [] [] [] [] [] [] []

Inconsistent. Inode bitmap should be 100000000.
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]] [ ] [ ] [ ] [ ] [ ]

Consistent

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] ] [ ] [ ] [ ] [ ]

No dir entry pointing to inode 1 or 2

Inconsistent
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

Fsck: 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.
1. Start transaction
2. Write blocks that belong to Tx to journal
3. End Tx.
transaction: write A to block 5; write B to block 2

Checkpoint: Writing new data to in-place locations

Transaction

Checkpoint

Free journal for next Txn
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

Journal on DISK!

write order 9,10,11
CHECKSUM OPTIMIZATION

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

write order before

9, 10, 11

write order after

9, 10, 11, 12

mark journal block as free
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: T1 T2 T3 T4

0 128 MB
Observation: Most of writes are user data (esp sequential writes)

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

For regular data, write it back whenever convenient.
METADATA JOURNALING

Before data block was written, crash!

Transaction: append to inode 1

Crash !?!

Garbage data

Not Correct!

Wrong Atomicity
Ordered Journaling

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

What happens if crash in between?

1. First data blocks written in place.
2. Barrier
3. Start Txn metadata
4. Txn End
5. In-place update for metadata

If data blocks are written and the crash as if nothing happened.
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
Blocks

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

- Bitmap
- Inode
- Journal

Write 5, 6
Write 8, 9, 10
Barrier
Write 11
Barrier
Write 4, 2
NEXT STEPS

Next class: How to create a file system optimized for writes

Project 5: Are you working on it?!