Practical Online Filesystem Checking and Repair

Daniel Phillips

Samsung Research America (Silicon Valley)

d.phillips@partner.samsung.com

Online filesystem checking

Why we want online checking:

- Fsck time at tens of minutes and heading higher
- SSD arrive and fsck algorithms got better
- But still an issue for:
  - Most laptops and workstations
  - Your bulk storage
  - Data centers

Everybody has had the fsck experience at just the wrong ti me.

Online filesystem checking

"Waiting for the next fsck"

Online filesystem checking

"You should strongly consider the consequences of disabling mount-count-dependent checking entirely. Bad disk drives, cables, memory, and kernel bugs could all corrupt a filesystem without marking the filesystem dirty or in error."

-- tune2fs man page

Online filesystem checking

What about checksumming?

- Easy to implement
- Relatively efficient
- Increases confidence in your data
- Not a substitute for repair!
- Does not tell you much about what is corrupted
- Eventually a checksum will certainly fail. Then what?

Online filesystem checking

Reasons for not attempting online checking:

- SSD makes the problem go away
  - 20 times smaller, 20 times faster = 400 times less annoying
- Handheld revolution changed the game
  - Phone reboots are relatively rare
- Hide the problem in the cloud
  - Let somebody else worry about it
- It’s not easy
  - We have more pressing issues
- Maybe it won’t happen to me

Online filesystem checking

It’s like changing a tire without stopping the car.

Online filesystem checking

Online filesystem checking

Reasons for doing it:

- Sweetens the availability equation for data centers
  - Ultimately saves money
- Longer practical uptime
- Even a small stall is annoying on a phone
- It’s a challenging engineering problem
  - The last big unsolved problem in storage?
- It’s a great research problem
  - Carve your initials in the history of storage technology!

Online filesystem checking

Online filesystem checking

The purpose of this work is to remove
"we don't know how to do it"
as a reason
for not doing it.

Online filesystem checking

Damage that really hurts:

- Block marked free when it isn't
  - Corruption propagates rapidly
- Multiple references to block
  - Becomes a double free
- Lose block of pointers high in the tree
  - Can lose the entire filesystem
  - A constant danger for CoW filesystem designs
- Disconnected directory tree
  - Where did my files go??

Online filesystem checking

How do you check a filesystem while it is changing?

- Frontend/Backend separation is a big help
  - Can still write to cache while backend suspended
  - Way better than freezing at VFS level
- It has to be incremental
  - Does not have to be fast
  - Must not kill performance
- Topology could change at any time
  - Algorithms must tolerate this
- Global structure much harder to check than local
  - From here on, just consider global checks

Online filesystem checking

Global structure we need to check

- Physical structure:
  - block leaks and double references
- Logical structure:
  - directory connectivity and loops
  - inode leaks and link counts

This is hard.

How do we do it offline?

Check physical structure

Data structure: Shadow bitmap

- One bit for each volume block

Algorithm:

1) Walk tree and mark off blocks in shadow

- Blocks already marked are double referenced

2) Compare bitmaps to shadow bitmaps

- Blocks free in shadow but not in bitmap are lost
- Blocks free in bitmap but not in shadow are future corruption

How do we do it offline?

Check logical structure

Data structure: Link count hash map for all inodes

Algorithm:

1) Walk directories in directory tree order
   - Increment entry target in hash
   - Already seen directories are loops
2) Walk inode table comparing link counts to hash
   - Unreferenced inodes are lost
   - Excess references are future double frees

Online filesystem checking

Online checking is **way** harder.

Let’s try to make it easier...

Online filesystem checking

Big idea: a new purpose for an old idea

- Block groups!
  - Already need them for allocation algorithms
- Introduce a “far map” of “far pointers” per group
  - Far pointers are supposed to be relatively rare
  - Allocation policy tries to enforce this
- Low level pointer changes update far map
  - Update far map only if destination changes
- High level filesystem algorithms otherwise unchanged

Online filesystem checking

More about far maps

Far map entry fields:

- Source group
- Destination offset
- Pointer type
  - Inode table tree
  - File data tree
  - Data extent
- Block Count

(Note: no source offset)

Online filesystem checking

Far maps are not just for online checking

- Many online features require reverse mapping
  - Online block migration
  - Online filesystem shrink
  - Online defragmentation
- Far maps are *not* reverse maps
  - ...but far maps enable reverse maps on demand
  - Much faster than persistent reverse maps

It is certain we need this new metadata. Now we can view online fsck as a “simple” extension.

Online filesystem checking

"Monotonic progress"... a recurring theme

Need stable enumerations:

- By block group
- By inode number

Examples of unstable enumerations:

- Filesystem tree order unstable for physical checks
- Directory order unstable for namespace checks

Online filesystem checking

The master plan:

1) Sweep bottom to top by block groups
   - Check block group and mark good
2) Sweep bottom to top in inode order
   - Check directory path to root and mark good
3) Clear checked bits and do it again.

Online filesystem checking

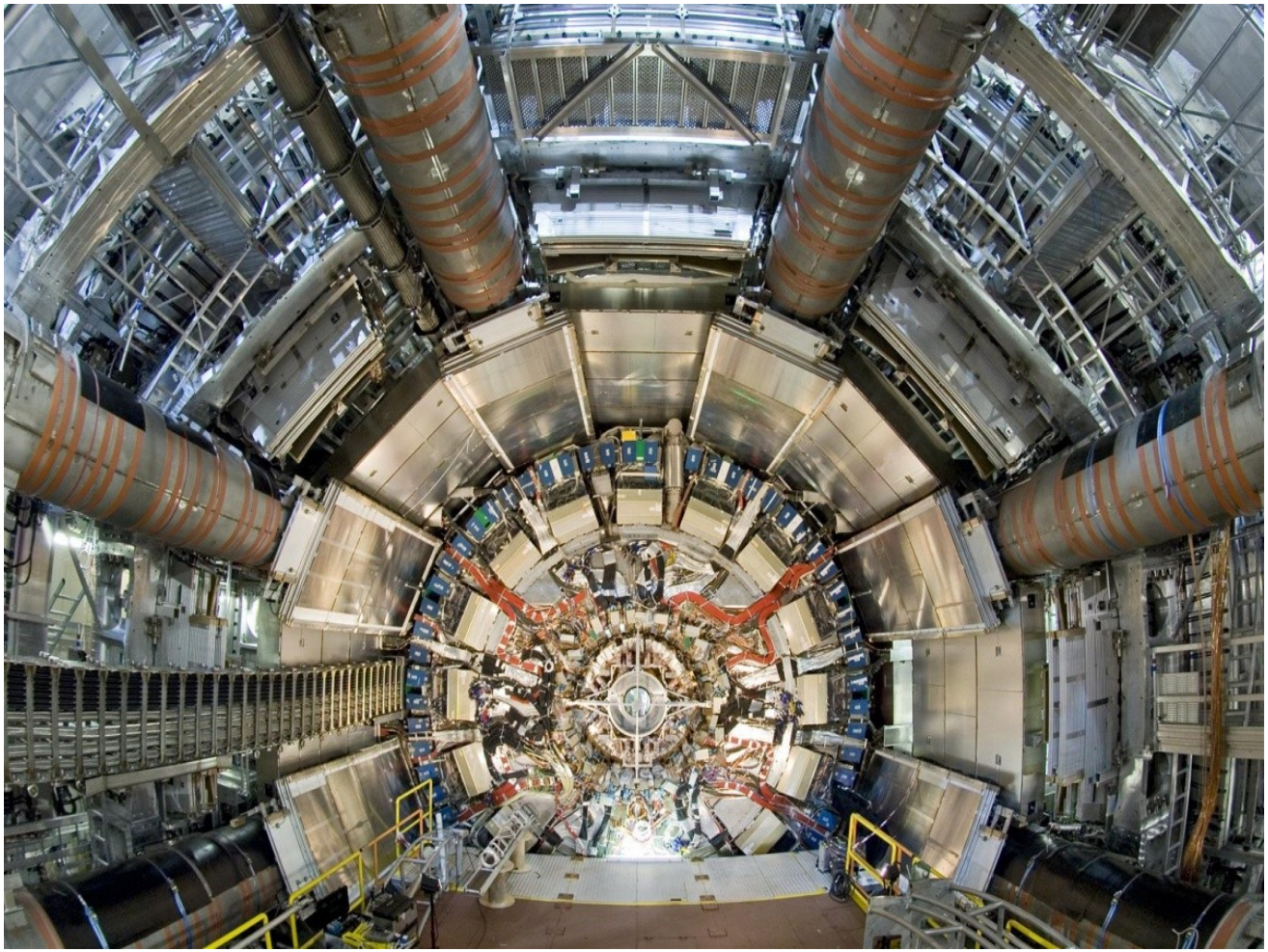
"Hermetic accounting" per group

- Inodes + far map entries $\Rightarrow$ local subtree roots
- Walk local subtrees $\Rightarrow$ shadow map
- If shadow map matches bitmap the group is good

What does that give us?

- Less than a second to read group into cache
  - Old school hard disk
- Usually no visible stall
  - Frontend updates continue in cache
- Already a nontrivial level of integrity checking

Online filesystem checking



the Atlas experiment

Online filesystem checking

A short break for some philosophy.

- We rely on far maps to do incremental group consistency

How do we know the far maps are consistent?

We don't. But what is consistency, really?

- An inconsistent system has ambiguous interpretations
- Choose the one that does the least harm

How do we know an updated group is still consistent?

We don't. Consistency has a shelf life.

- A group marked good just means we trust it more
- We trust far maps if we can read them
- Great place for a checksum!

Online filesystem repair

Rebuilding far maps online

- Start with all empty far maps
- Live far pointer updates can continue
  - Do not complain about missing ones
- Walk itable per group
  - Inode number to group correspondence
- Rely on inode structures, bitmaps and partial far maps
  - Expect non hermetic accounting
  - Any used blocks not accounted for are "in the fog"
- Keep a map of fogged regions by group
- Eventually find a far pointer to earlier fog region
  - Recursively resolve other fogged regions

Online filesystem checking

Incremental directory connectivity and loops

- Can’t walk in directory order because tree can mutate during walk
- Walk in inode order instead
  - For each directory inode, walk parent pointers to root
    - To find loops, keep hash of directories already seen
    - Stop at first directory marked good
  - At each step, verify parent references child
    - Store “name attribute” per directory
- Not as efficient as directory order walk, but we have more time because it’s online

Online filesystem checking

Inode reference counts and leaks

- Walk in inode table order
- Update hash of inode reference counts
- Mirror link count updates to hash when source inode lies below walk cursor
- Otherwise, just like the offline check

Online filesystem repair

Repair is the easy part.

- The hard part is deciding what is broken
- Helpful cache model: Tree rooted in dirty cache
  - Repair the *cache view* of the filesystem
  - Commit delta when things look good
- Prefer to trust structure over bitmaps
  - Be quick to mark referenced blocks used
  - Be slow to mark unreferenced blocks free
- Recreate missing inodes from open files
- If things get ugly we can always go offline

Online filesystem repair

Low level freespace scan

- Directory was lost and user wants data back!
- We should probably go offline, but...
  - Why not try hunting through free space and see what we can find?
  - Caveat: our running system might overwrite what we are looking for
- OK, it looks bad, let’s just go offline
  - Then we need to deal with...

Online filesystem repair

litter

Online filesystem repair

Which one is the real one?

Online filesystem repair

Litter

- Copy on write filesystem designs all create litter
- Fixed metadata position like Ext4 is a big advantage
- Introduce “uptags” that we can scan for:
  - Magic number type tag
  - Delta counter
  - Owner
  - Logical position
- To save space, just store low order bits
  - This is all about improving probabilities

Online filesystem repair

Conclusion

- Add a little metadata, get a lot of results
- New far map metadata resembles reverse map
- Runtime overhead looks small
- Even basic physical checks are already useful
- Repair is easy if we know what is wrong
- Desktop, servers and data centers benefit

The only thing standing in the way: a lot of hard work

Questions?

Daniel Phillips

Samsung Research America (Silicon Valley)

d.phillips@partner.samsung.com