Recon: Verifying File System Consistency at Runtime

Daniel Fryer, Kuei Sun, Rahat Mahmood, TingHao Cheng, Shaun Benjamin, Angela Demke Brown, Ashvin Goel

Department of Computer Science, University of Toronto
Department of Electrical & Computer Engineering, University of Toronto

Data in Danger!

Metadata corruption bugs continue to be found in commodity file systems!

- Checksums & redundancy don't protect against errors originating within the file system itself
- Backups could be corrupted, stale
- *fsck* is slow and requires the system to be offline
- N version systems have high overhead, limited features

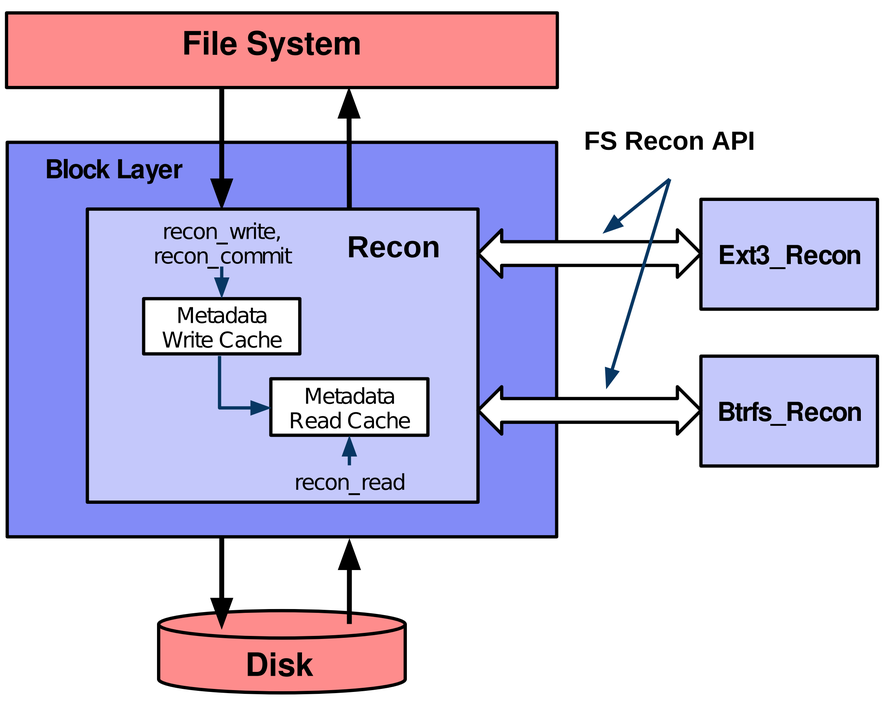
Goal

Detect corruption before it reaches the disk – protecting the file system from itself!

Key Insight

We can take advantage of the existing transaction model used to enforce crash consistency

Recon Architecture



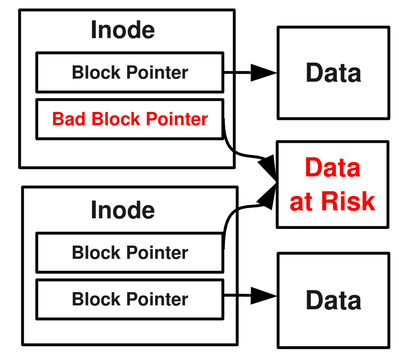
Approach

- Monitor all I/O between file system and disk
- Check the same consistency properties as fsck - but at run time
- Transform slow global checks into fast, local checks
- Operate outside of the file system
- Use induction: prove consistency after a transction commits, given prior consistent state
- On failure, several options available

Example: Protection against double block allocation

Global check (across entire disk):
Each block pointer is unique and matches the block allocation bitmap

Local checks (within a single transaction):
Match block pointer changes to bitmap changes, and ensure new pointers are unique within transaction

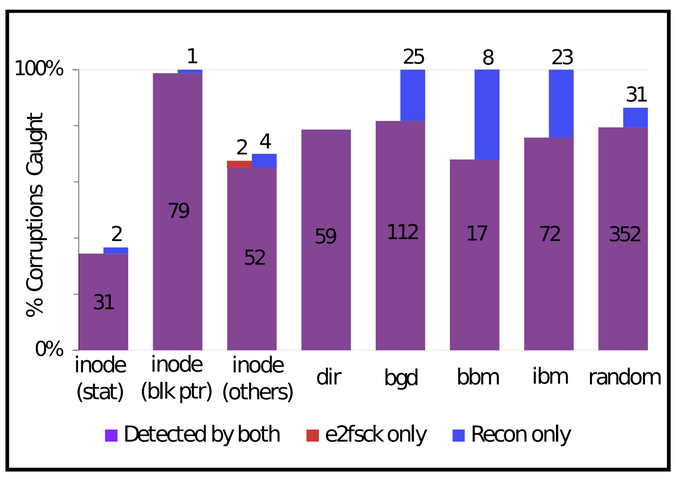


Corruption example: double block allocation

Future Work

- Go beyond *fsck* Can we enforce operation specific constraints?
- Generalize to more file systems
- Explore application level invariants
- Use hypervisor to protect checking mechanism
- Implement invariant checking in a declarative language

Recon vs. fsck



Catch rate for random corruption close to *fsck*

Recon automatically checks unused areas ignored by fsck

Low performance impact!

Recon	fsck
Run-time	Offline
Checks metadata before writing to disk	Checks metadata after writing to disk
Protects data from corruption	Repairs corrupt data (partially)
Assumes consistent initial state	**No assumptions** about initial consistency

Overhead

Performance (256MB Cache, 128MB Journal)