Protecting the File System From Itself

Daniel Fryer, Angela Demke Brown, Ashvin Goel

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

Problem

- Even stable file systems have bugs
- File system bugs can corrupt data
- Checksums and replication are no defense against FS bugs
- N-version systems are expensive
- Tools like **fsck** try to repair damage after the fact
  - Too slow (offline!), data loss still possible
  - Some problems can't be caught by offline checks

Key Idea

Check transactions against invariants before committing to disk

Transactional Invariants

- Describe the behavior of correct transactions
- Ensure on-disk structure of file system remains valid
- Hold after each atomic operation

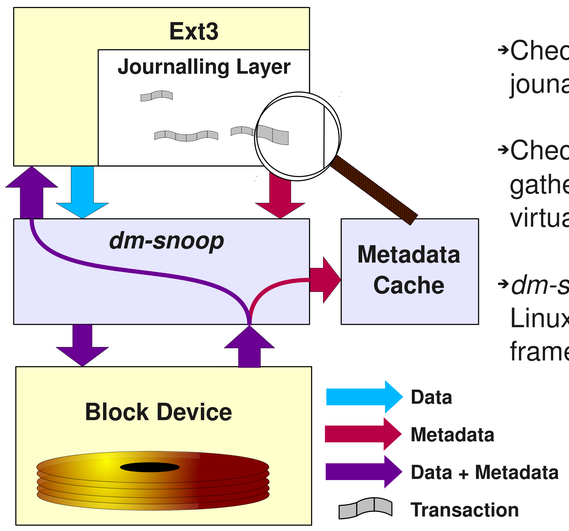
Research Questions

- What kinds of invariants can we check quickly?
- How do we specify invariants?
- How thoroughly can we specify file system correctness?

Current Prototype

Strategy for developing prototype

- Choose known, reproducible, data-corrupting Ext3 bugs
- Identify violated invariant
- Implement checking function for invariant



- Checking is done in the Ext3 jounalling layer (jbd)
- Checking relies on metadata gathered by *dm-snoop*, a virtual block device
- *dm-snoop* implemented using Linux "device mapper" framework

Challenges

How are invariants specified?

- Declaratively (e.g. as in SQCK)

How do we maintain consistency?

- Assume that file system is consistent before the transaction
- Prove that it will be consistent afterwards

What is needed to verify an invariant?

- Metadata – likely to have been read recently
- Cache necessary to avoid extra reads

Handling Failures

Return error

- Allows an application to retry operation or find an alternative
- Doesn't work if application believes that the transaction has succeeded

Snapshot FS and continue

- Preserves all data
- Defers recovery until future "convenient" time
- Feasible to implement at block layer

Example: Ext3 Invariants

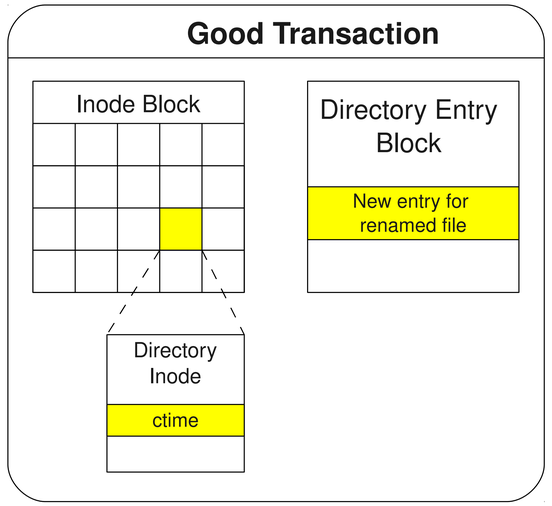
- Data block in use
  <=> Block Bitmap bit is set
  <=> Block number appears in exactly 1 inode, indirect block, 2-indirect block or 3-indirect block
  <=> Block is dir. entry, indirect, 2-indirect, 3-indirect or file data
- When a file data block is written to, the ctime field in its inode should be updated
- Data is only written to allocated blocks (Not so true if data is not journalled!)

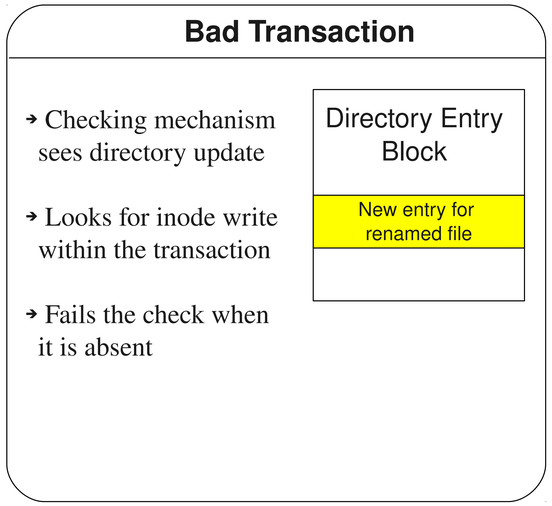
Example Bug: "Directory ctime not updated on rename"

- Bug #10276 on bugzilla.kernel.org
- Use of the "rename" operation could move a file into a directory without updating the directory's creation/ modification time field.

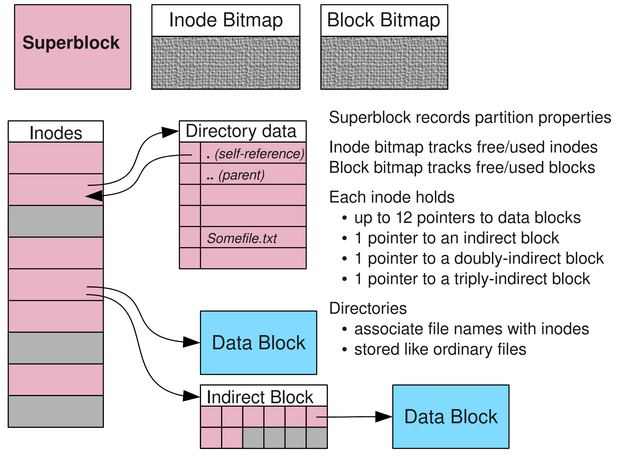
`fsck` can't catch or repair this problem!

- Dates may be changed deliberately afterwards
- File may be older than the directory that contains it





Important data structures in Ext3



Superblock records partition properties

Inode bitmap tracks free/used inodes
Block bitmap tracks free/used blocks

Each inode holds
- up to 12 pointers to data blocks
- 1 pointer to an indirect block
- 1 pointer to a doubly-indirect block
- 1 pointer to a triply-indirect block

Directories
- associate file names with inodes
- stored like ordinary files