File System Failure : Recovery

04/29/2021

Professor Amanda Bienz

Pages 586-592

Recovery

- **Consistency checking :** compare data in directory structure with data blocks on disk, and try to fix inconsistencies
  - Can be slow and sometimes fail
- Use system programs to **back up** data from disk to another storage device (external hard drive, cloud)
- Recover lost file or disk by **restoring** data from backup

Journaling (1)

- Journaling (Write-Ahead Logging)
  - When updating disk, before overwriting structures in place, first write down a little note describing what you are about to do
  - Writing this note is the “write ahead” part, and we write it to a structure that we organize as a “log”
  - By writing note to disk, you guarantee that if a crash takes place during update of structures, you can go back and look at the note you made and try again
  - Thus, you will know exactly what to fix after a crash, instead of having to scan the entire disk

Journaling (Cont.)

- How Linux ext3 incorporates journaling into the file system:
  - Most on-disk structures are identical to Linux ext2
  - New key structure is the journal itself
  - It occupies some small amount of space within the partition or on another device

Fig.1 Ext2 File system structure

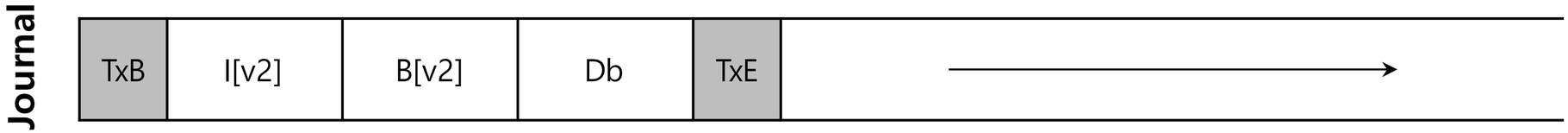
Fig.2 Ext3 File system structure

Data Journaling (1)

- Data journaling is available as a mode with the ext3 file system
- Example : our canonical update again
  - We wish to update inode (I[v2]), bitmap(B[v2]) and data block (Db) to disk
  - Before writing them to their final disk locations, we are now first going to write them to the log (a.k.a. journal)

Data Journaling (2)

- Example : our canonical update again (Cont.)



  - TxB : transaction begin block : it contains some kind of transaction identifier (TID)
  - Middle three blocks just contain exact content of the blocks themselves
    - This is known as physical logging
  - TxE : transaction end block : marker of the end of the transaction, also contains the TID

Data Journaling (3)

- Checkpoint
  - Once this transaction is safely on disk, we are ready to overwrite the old structures in the file system
  - This process is called checkpointing
  - Thus, to checkpoint the file system, we issue the writes I[v2], B[v2], and Db to their disk locations

Data Journaling (4)

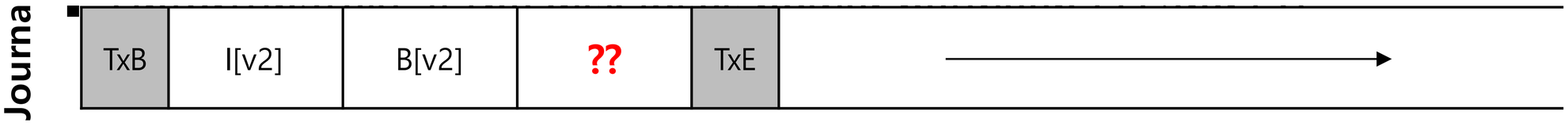
- Our initial sequence of operations:
  - Journal write:
    - Write the transaction to the log and wait for these writes to complete
    - TxB, all pending data, metadata updates, TxE
  - Checkpoint
    - Write the pending metadata and data updates to their final locations

Data Journaling (5)

- When a crash occurs during the writes to the journal
  - Transaction each one at a time
    - 5 transactions (TxB, I[v2], B[v2], Db, TxE)
    - This is slow because of waiting for each to complete
  - Transaction all block writes at once
    - Five writes -> a single sequential write : faster ways
    - However, this is unsafe
      - Given such a big write, the disk internally may perform scheduling and complete small pieces of the big write in any order

Data Journaling (6)

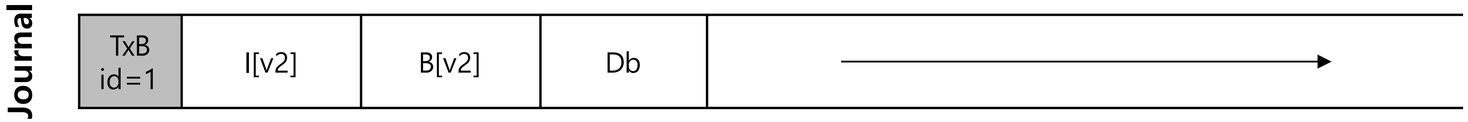
- When a crash occurs during the writes to the journal (Cont.)
  - Transaction all block writes at once (Cont.)
    - Thus the disk internally may (1) write TxB, I[v2], B[v2], and TxE.
      (2) Later, write Db
    - What if disk loses power between (1) and (2)?



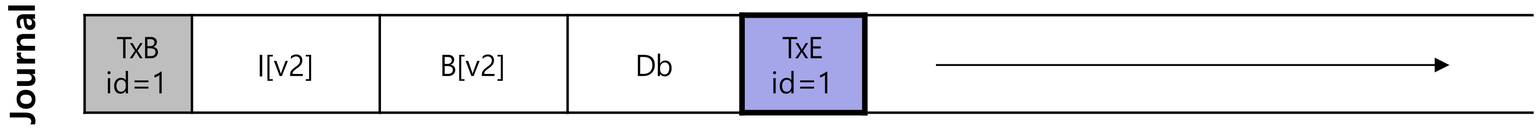
  - Transaction looks like a valid transaction
    - Further, the file system can’t look at the fourth block and know it’s wrong
    - It is much worse if it happens to a critical piece of the file system (i.e. superblock)

Data Journaling (7)

- When a crash occurs during the writes to the journal (Cont.)
  - Transaction all block writes at once (Cont.)
    - To avoid this problem, the file system issues the transactional write in two steps
    - First, write all blocks except the TxE block to journal



    - Second, issue the write of the TxE



    - Important aspect of this process is atomicity guarantee provided by the disk.
      - Guarantees that any 512-byte write either happen or note, thus TxE should be a single 512-byte block

Data Journaling (8)

- When a crash occurs during the writes to the journal (Cont.)
  - Transaction all block writes at once (Cont.)
    - Thus, our current protocol to update the file system with each of its three phases labeled
      - Journal write: write contents of transaction to the log
      - **Journal commit (added)** : write transaction commit block
      - Checkpoint : write contents of the update to their locations

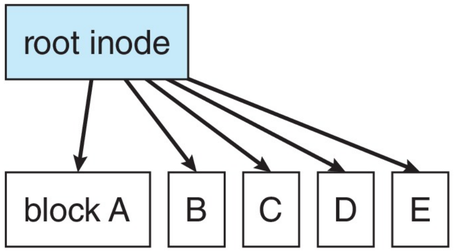
Data Journaling (end)

- Recovery
  - If the crash happens before the transactions are written to the log
    - Pending update is skipped
  - If crash happens after transactions are written to the log, but before the checkpoint:
    - Recover the update as follows:
      - Scan the log and lock for transactions that have committed to the disk
      - Transactions are replayed

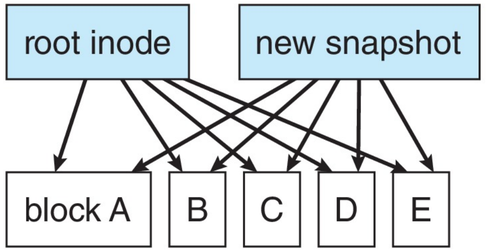
Other Solutions

- Never overwrite blocks with new data
  - Transaction writes all data and metadata changes to new blocks
  - When transaction is complete, metadata structures that pointed to old versions of these blocks are updated to point to new blocks
  - Could remove old pointers and blocks, or keep then to create a **snapshot** : a view of the file system at a specific point in time (before any updates after that time were applied)
  - If pointer update is done atomically, no consistency checking is necessary
- WAFL : write anywhere file system, snapshots and consistency checking

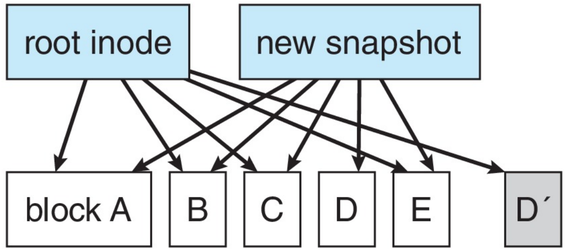
WAFL



(a) Before a snapshot.



(b) After a snapshot, before any blocks change.



(c) After block D has changed to D´.

Backup and Recovery

- System programs can back up data from one storage device to another
- Recovery from loss of file (or entire device) may then only be a matter of restoring data from the backup
- Minimize copies : use information from each directory entry (compare last modified with last backup time and only update necessary files)

Typical Backup Schedule

- Day 1 : copy all files from file system to backup (**full backup**)
- Day 2: copy all files changed since day 1 to a second location
- Day 3: copy all files changed since day 2 to third location
- … Day N : copy all files changed since day N-1. Then go back to day 1
- **Can restore entire file system, but can also restore a file that was accidentally deleted at some point**

That’s All Folks

- Next Tuesday, review for test 4
- Next Thursday, review for optional test 5, or feel free to take test 4 during class