Lecture S6: Write-anywhere file systems

****
Review -- 1 min
****

Naming: name->filenum
file num -> data blocks

****
Outline - 1 min
****

Write anywhere file systems
[TBD]

****
Preview - 1 min
****

****
Lecture - 20 min
****

1. Transactions in file systems

1.1 write-ahead logging

Almost all file systems built since 1985 use write-ahead logging (windows NT, solaris, OSF, Linux JFS, SGI XFS, etc)
Idea: write all changes in a transaction log (update directory, allocate blocks, etc) before sending any changes to disk
→ "create file", "delete file", "move file" etc are transactions

eliminates need to "fsck" after crash

If crash
- read log
  - if log isn't committed, no change
  - if log is comitted, apply all changes to disk
  - if log is zero, then all updates have gotten to disk

Advantage:
- + reliability
- + asynch write-behind (seeks)
- DA: all data written twice (→ often, only log metadata)

1.2 Log-structured file system

Idea: write data only once by having log be only copy on disk
→ as you modify disk blocks, just store them out on disk in the log.
Put everything: data blocks, file headers, etc. on log

If need to get data from disk, get it from the log

- can store data blocks, indirect blocks, etc anywhere on disk, so no problem to put in log
- put inodes in log, too

→ need some way to find them

- imap is array of pointers to inodes

inodes no longer in fixed location, but imap is in fixed location (actually two fixed locations called "checkpoints")

"apply changes to disk" now means update on-disk imap
"replay log after crash" now means apply changes of comitted transactions to imap.

Advantages

- all writes are sequential!

No seeks, except for reads, but

- RAM getting bigger → caches getting bigger
  - in extreme case (infinite cache) → disk I/O only for writes (only for durability of data)

→ conclude, optimize for writes. LFS does that

Cleaning

Eventually, log wraps around – run out of room

→ have to garbage collect.

Majority of files deleted within first 5 minutes, so go back over log, and compress pieces that are no longer in use

As disk gets full, need to clean more frequently, so keep disk under-utilized

Pros & cons

+ write performance

+ read performance (if write order predicts read order)

■ cleaning cost (off-line?)

■ bad if disk full, random updates to files

SEE
http://www.cs.utexas.edu/users/dahlin/Classes/GradOS/lectures/lfs.pdf

****

Summary - 1 min

****

Key idea: ...