LECTURE

LFS

Perf: FFS (main idea?)
block groups, etc.

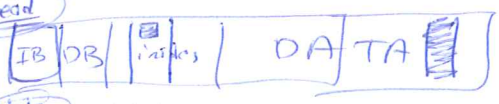
Crash consistency: Journaling
main idea?

Today: Perf + Crash Consistency ⇒ LFS

Problem: (FFS perf good, not great)
e.g. create a 1-block file
(what structures are read, written?)

read dir inodes
read dir data (see if fname is unique)

(1) all the blocks you must read
(2) all the blocks you must write



block group
read inode bitmap (find free)
write inode bitmap (mark allocated)
write dir data (name → new inode num)
write dir inode (update length)
read inode block (mark allocated)
write inode block (now initialized)

read data bitmap
write " "
write data too

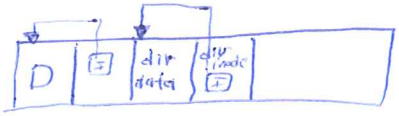
what happens w/ cache? (as it gets larger?)

cache is central shaper of traffic
reads: avoided
writes: delayed (+scheduled)

conclude: Perf of ~~FFS~~ under lots of file creates: localized, but still lots of (seeks/rotates)

real goal: use disk (purely sequentially)
hard for reads → why? if file x, y are not near each other...
easy for writes → why? can always write to unallocated space

let's pretend: all reads are in cache
how to write sequentially?



⇐ in memory

segment (say 1MB)

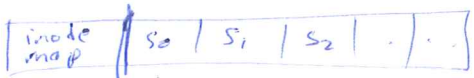
Protocol: ~~attempt~~ #1:
(1) put data, inodes into in-memory segment
(2) when full, write to disk
disk: just a bunch of (empty segments)

Problem: open("/foo", O_RDONLY)
typical fs:
read root inode
read root data (find foo)
read foo inode

given inode #, how to find? (easy: calculate)
inode

LFS: how to find inode?
1) scan disk
2) indirection: (inode map)
array: indexed by inode #
content: addr of latest copy of inode

protocol #2: inode map @ fixed location

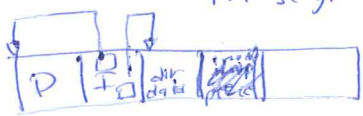


millions of entries (4B each)

(1) fill segment (in memory)
update inode map (in memory)
(2) write segment
write dirty parts of inode map

but: problem? writes not that sequential anymore

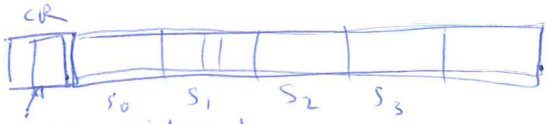
protocol #3: inode map pieces in segments!



(1) fill (2) write ⇒ purely sequential

problem? can't find inode map

protocol #4: checkpoint region



includes pointers to pieces of inode map

(1) fill blocks, update inode map (in memory)
(2) write segment (seq.)
(3) occasionally (every 30 secs) checkpoint by updating CR w/ inode map

great freedom for writes (write anywhere)

what if crash?

problem ① between CR update
② during CR update

Review: how to open/read a file?

assume: all on disk

CR → inode map → inode → {file ...}

once here, save

really: on boot, read in inode map, cache it

⇒ no slower

influence:

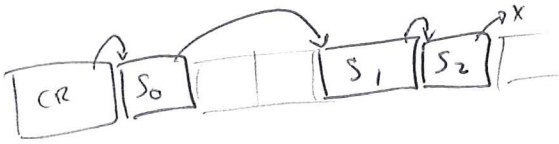
NetApp (wafl)

ZFS (cow)

new problem: disk gets full

what to do? is disk really full?

what about crash recovery?

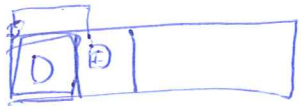


⇒ roll forward

cleaner (garbage collector)

→ goal: find dead blocks, make available for reuse

mechanism: how to determine if dead?



cleaner

① read in segment

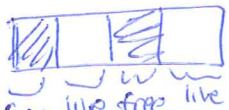
need:
- inode # of data block
- offset in file

② how to determine if inode is live? (easy: imap)

how to determine if data is live? (easy: check live inode's map)

then: Check if live inode ⇒ (pts to) this data block

result



⇒ problem: can't write sequentially

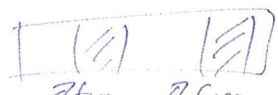
solution:

read in N segments ~~blocks~~, write M like segments ~~blocks~~

(M < N)

free N old segments

problem: how big to make segments



seek, but...

problem: which blocks to clean?

random:

hot/cold: } policy