FFS

(Fall '03)

→ Intro

→ Old FS + Problems

→ FFS : Cylinder Group + Big Blocks

→ Global allocation
(+ large-file exception)

→ Dealing w/ fragments

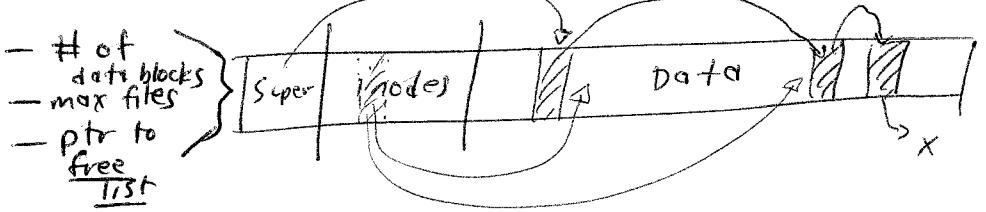
→ Wrap up
Parameterization, Performance, other enhancements

FFS: Technology-aware file system

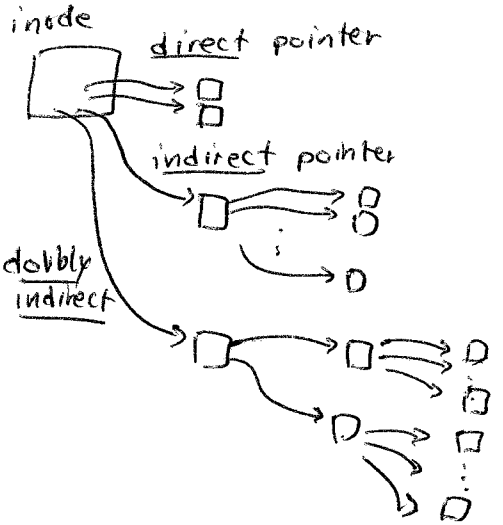
Problem: Old FS ⇒ Poor performance
not aware of technology

Solution: Better data layout
"group related things", "use big blocks"
(+ some new functionality)

"Old" File System



Review: inode



Observations

→ Long distance between inodes / data
→ inodes in single directory not close to one another
→ small blocks (512b)
→ blocks laid out poorly
→ free list ⇒ scrambled (random allocation) ⇒ seeks

Result: 2% of potential b/w !
(and worse over time)

Larger blocks: better, but not the solution

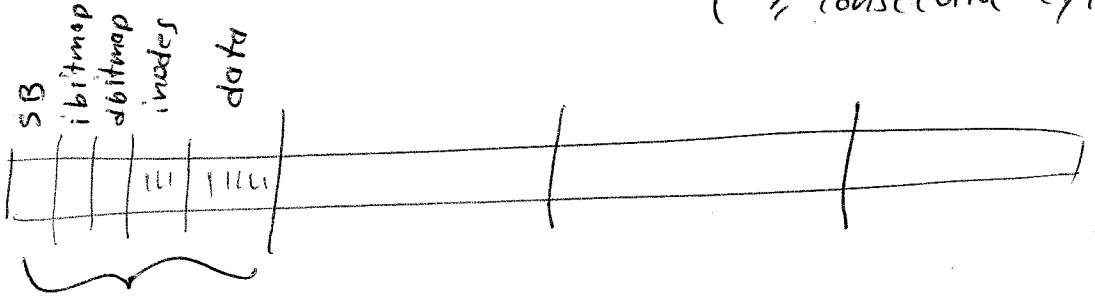
Solution: the Berkeley Fast FS

Technology aware: it's a disk, not memory!

⇒ "spatial locality" exists

Basic structuring concept: cylinder group

(≥ consecutive cylinders)



CG

- ⇒ copy of super block
- ⇒ bitmaps (not free list) inodes / data
- ⇒ inodes
- ⇒ data

Also, use big blocks : 4 KB

⇒ good performance, but ...

⇒ waste (internal fragmentation) [~50% of disk]

Questions:

- ⇒ How to place data on disk?
- ⇒ How to use big blocks + yet avoid wasting space?

Allocation Policy

Global:

Goal: cluster related info
(corollary: spread out unrelated)

Must think about: files, directories
basic user structures for interacting w/ FS

Assumptions:
⇒ Files in directory accessed together (e.g., ls ..., compilation, ..)] and convene
⇒ File: inode + data accessed together (esp. true for writes)

Result:
⇒ mkdir (create directory)
pick CG w/ high # of free inodes, low # of directories

⇒ creat (create file)
for all inodes in dir. D, place in same CG
for all data of file, place in same CG as inode

Exception: Large Files
Why? ⇐ [when file > threshold, chain to new CG
⇒ how to pick threshold?
[cost of seek vs. transfer size]

Local policy:
Global picks block, but does not have perfect info
Local finds exact block: next rot in cyl, next in CG, hash to CG, exhaustive

Dealing w/ Big Blocks

⇒ Big blocks ⇒ waste (fragmentation)
but, want for higher performance

⇒ Solution: sub-blocks

⇒ When used? Allocation of new data
(when is this done? on write() sys call)

⇒ How?
- ⇒ if space for new exists in allocated,
  put there
- ⇒ if no space for new + no existing fragments,
  fill (n-1) blocks, n^{th} ⇒ block or fragment
- ⇒ if new doesn't fit in fragments,
  and file has fragments,
  and new + fragments > block,

Problem? (expensive) { copy frags + new data into new blocks, free fragments

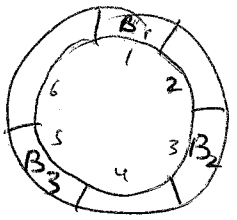
How to avoid copy?
write block at a time

⇒ Result: { big-block performance, low waste }

Other issues:

> Parameterization

Position blocks well rotationally



⇒ why? (to avoid missing [no DMA] block)

⇒ why bad?

1/2 the bandwidth

(but better than rotations)

⇒ not needed today

(disks handle automatically)

Performance:

> instead of 2% of B/W, 50%

(and robust over time)

Enhancements

> long file names, symbolic links, atomic rename, quotas

Summary

File system: on-disk data structure w/ certain common usages

FFS: understand common usage
understand technology
build "disk conscious" data structure

Q) What did they miss?