

# FFS

CS380L: Mike Dahlin

October 9, 2007

“Treat disks like tape.”

*J. Ousterhout*

## 1 Preliminaries

### 1.1 Review

### 1.2 Outline

- Overview of file systems – trends, future
- Disk drive modeling
- FFS

### 1.3 Preview

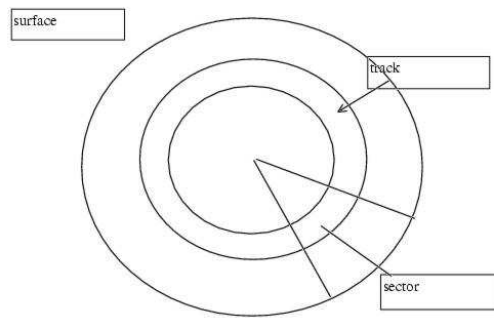
(see below)

## 2 Overview of file systems

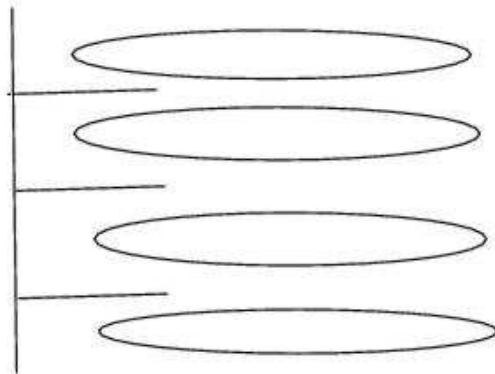
- Phase 1: Basic file systems (e.g., Unix pre-FFS)
  - Naming: name  $\rightarrow$  fileID  $\rightarrow$  blocks
  - Allocation: free list, bitmap, ...
- Phase 2: Performance/efficiency/locality (e.g., FFS)
  - Technical motivation: seek times, rotation times v. transfer times; capacity increases faster than performance

- Workload motivation: locality (sequential within files, locality within directories)
- Approach: locality within files, directories; waste space for performance
- More today: *A Fast File System for Unix*
- Phase 3: Optimize writes – reliable, efficient (e.g., JFS, LFS)
  - Workload motivation: *don't lose data*
  - Technical motivation: (1) big memory → writes dominate reads (?), (2) ordering writes via careful ordering of updates → lots of sequentially-constrained seeks
  - Approach: Transactions + logging; almost all modern file systems are “journaling” – metadata updates are done via transaction, data updates done in place
  - More next week: *The transaction concept, LFS*
  - **Guest lecture: IBM Linux Technology Center “JFS for Linux” and “Linux Volume Manager” – W Oct 23 5-6:30PM TAY 2.106**
- Phase 4: Scalable, distributed (e.g., RAID, XFS, xFS, NASD, ...)
  - Technical motivation: (1) networks → can concentrate more users (e.g., from across enterprise) on a file server; (2) networks → can distribute file system across many disks/servers; (3) cheap silicon → can put more smarts into disk
  - Approach 1 (widely adopted): RAID, scalable FS (XFS)
  - Approach 2: (research systems) – smart disks, xFS, NASD, Frangipani, ...

### 3 Disk drive modeling



- Disk *surface*: circular disk, coated with magnetic material
- *Tracks*: concentric rings around disk surface, bits laid out serially along each track
- Each track is split up into *sectors*: arc of track; also, minimal unit of transfer
- Disks spin continuously
- Disks organized as set of *platters* in a stack



- Disk is read/write via a comb - 2 read/write "heads" at end of each "arm"
- *cylinder* - corresponds to track on each surface

- Disk operation is in terms of radial coordinates (not x, y, z) - move arm to correct track, wait for disk to rotate under head, select head, then transfer as it is going by

High end disk today: Seagate Cheetah ST373405LC (March 2002)

Capacity	73GB
# surfaces per pack	8
# cylinders	29,549
total # tracks per system	236,394
# sectors per track	776 (avg)
# bytes per sector	512
# revolutions per minute	10000
buffer size	4MB
xfer rate	50MB/s-85MB/s (internal) 38-64 MB/s (internal, formatted) 160MB/s (external, max)
seek time	avg: 5.1ms (r) 5.5ms (w) 1 track: .4ms (r) .6ms (w) full disk: 9.4ms (r) 9.8ms (w)
idle power	10W
MTBF	1,200,000 hours
Nonrecoverable read errors	1 per $10^{15}$ bits read

Note: nonrecoverable read errors may be becoming significant: 1GB =  $10^{10}$  bits; 128GB =  $10^{12}$  bits. Based on worst-case guarantees by manufacturers, nontrivial (1/1000?) probability that you will successfully read an entire disk. And, nonrecoverable read errors are improving slowly (Pete Chen's 1994 RAID survey cites a figure of 1 per  $10^{14}$  in 1994 - disks have increased in size by about 3 orders of magnitude while bit errors have improved by only 1 order of magnitude. This implies probability of this error has increased by 2 orders of magnitude over the last decade.

Possible project: examine this trend in more detail. Do we need to redesign single-disk file systems to use more redundancy? Or, should we count on disk manufacturers to back off on density and/or increase robustness of coding to keep this issue acceptable. Will everyone that cares about reliability use RAID? (What about laptops and other portable devices?)

### 3.1 what you need to know about disks

1. tech trends
2. failures

### 3. performance

#### 3.2 Tech trends

Disk density (dollars/byte) improving much faster than mechanical limitations

- cost per byte improves by 100% per year
- bandwidth improves by about 20-30% per year (10x per decade)
- seek, rotation get better by 5-10% per year (2-3x per decade)

Other key tech trend: form factor

- innovator's dilemma: 8 inch, 5 inch, 3.5inch, ...
- Today on 3.5 inch v. 2.5inch cusp (incumbants likely to survive)
- solid state drives are threatening (incumbants at risk)

#### 3.3 Reliability

Disks are supposed to store data permanently  
problems

- Block writes v. higher-level writes: transactions (next lecture)
- Disk crash
  - MTTF (advertised) O(1M hours – about 100 years)
  - MTTF valid during disk lifetime (about 5 years) – bathtub curve
  - In practice, expect to lose a few percent of your disks each year
- Block corruption
  - nonrecoverable read error (advertised) – 1 per  $10^{15}$  bits read (→ about 1 bad block per 10 TB...not ignorable any more!)

### 3.4 Disk performance

To read or write a disk block

0 *controller time*: Wilkes: about 1ms

1. *seek*: position heads over cylinder (+ *select head*)

- Time for seek depends on how fast you can move the arm.
- Typically  $\tilde{10}$ -20ms to move all the way across disk
- Typically 0.5-1ms to move 1 track or select different head on same cylinder
- Wilkes: small seek time is  $K0 + K1 * \sqrt{\text{ntracks}}$ , large seek time is  $K2 + K3 * \text{ntracks}$ 
  - HP 97560:  $3.24 + .400\sqrt{d}(d < 383)$   
 $8.00 + .008d(d \geq 383)$

2. *rotational delay*: wait for sector to rotate underneath head

- 10000 RPM - 166 revolutions per second  $\rightarrow$  6 ms per rotation

3. transfer bytes

- 0.5 KB/sector \* (100-1000)sectors/revolution \* 166 Rev per second = 8-80 MB/s
- (e.g. 10 MB/s  $\rightarrow$  .5KB sector  $\rightarrow$  0.005ms)
- overall time to do disk I/O: controller time + seek + rotational delay + transfer
- Question: are seek and rotational times overhead, latency, or bandwidth?
  - Seek and rotational delay are latency to CPU, overhead to disk
  - transfer rate is BW
  - controller time is latency to CPU, overhead to controller
- Random access
  - avg seek  $\tilde{4}$ ms
  - $\frac{1}{2}$  rotation  $\tilde{3}$ ms
  - transfer  $\tilde{0.02}$  ms

- → about 7ms to fetch/put data; mostly seek/rotation
- → 73 KB/s for random sector reads
- *note*: “random access” is pessimistic worst case – assumes no locality at all (assumes seek is 1/3 of distance across disk)
- Sequential access
  - What if next sector on same track? Then no seek, no rotation delay → 10+ MB/s
- **Key to using disk effectively (and therefore to everything in file systems) is to minimize seek and rotational delay**
  - Ousterhout: treat disks like tape
- Performance bottom line
  - performance standard model
    - \* seek + rot + xfer is standard model
    - \* seek – 4-10ms “average”
    - \* rot – 6-10ms full rotation , 3-5ms average
    - \* xfer – 40-80MB/s – .005ms per sector
  - gotchas
    - \* “average rotation” misnamed
    - \* rot > seek >> xfer
    - \* “break even” at 1MB xfer (ignoring limits on track size!)
    - \* standard model ignores track buffer
    - \* no such thing as a cylinder
    - \* stateful – need to model current head position, rot position, track sparing, etc to accurately model disk (see Ruemmler and wilkes)

## 4 Admin

- Exam: tuesday
  - In class
  - open book
  - Be ready for all papers.

- Projects: Checkpoint Oct 28 (3 weeks) – Think through the rest of the semester.

## 5 FFS

### 5.1 “Old” Unix FS

- Small blocks
  - 0.5-1K
- Files not layed out contiguously
  - Simple free list implementation
  - Fragmentation over time → need to study file system layout over time, not in lab
- Inodes far from data
- Inodes in a directory not together

### 5.2 FFS

- Solution part 1: Big blocks
  - 4-8K blocks
    - \* But most files are small → internal fragmentation
  - “Fragments” within blocks (.5-1K)
    - \* File size < block size: a number of contiguous fragments
    - \* File size > block size: a number of blocks plus a number of contiguous fragments
  - Solved?
    - \* Pretty good bandwidth for large files
    - \* Pretty good space utilization for small files
  - **Question:** Why don’t we have a 1MB block size and 1KB fragment size?
- Solution part 2: Inter-block locality
  - Organize freelist into cylinder groups – try to allocate within a group; search for new block:

- \* Rotationally closest in current cylinder
- \* Current cylinder group
- \* Another cylinder group
- Challenge: can't put everything next to everything
  - Each new directory goes to new cylinder group (localize inodes)
  - Start allocating blocks for files in directory's cylinder group
  - Move big files to different cylinder group before they fill it up
- Performance
  - 10x improvement
  - CPU, memory-to-memory copy are limiting factors
  - Optimal rotational positioning hard
- Criticisms
  - Heuristics and magic numbers: hack or science?
    - \* Paper skips interesting science: 90% heuristic; cylinder group concept; bigger blocks; ...
  - How will technology trends change these heuristics?
    - \* Do cylinder groups make sense any more? (Do cylinders?)
  - How will other workloads interact with heuristics?
    - \* Assumes directory is unit of locality...
    - \* What if patterns change?

## 6 P

erformance model

Questions

- Suppose you blanked out the experimental section. What experiments should they run? (What are they/should they be trying to show?)
- Suppose you blanked out the graphs, themselves. Is the system and methodology described well enough that you can develop a performance model to predict system performance?

Disk parameters: They don't provide all of the details about their disk performance. From looking on line, here is the info I was able to find on the AMPEX Capricorn 330-mb winchester drives used in this study.

- 32 sectors/track
- 16 tracks
- 1024 cylinders
- 3600 rpm

(Note that these numbers are not quite consistent with the paper – they come out to a 262MB disk rather than 330. Hopefully they are close enough to make sense of the results.)

## 6.1 P

erformance model

- What are goals/claims/metrics?
- Design an experiment to test each.
- Based on performance model of “old” FS, “ideal” FS, and “FFS” predict performance for these three cases
  - First pass: Predict shape of lines, which line above which other line
  - Second pass: predict actual values
- What is performance model for “old” FS?
  - *1024-byte blocks*
  - *Random layout of blocks (due to free space fragmentation + linked list free list) – each read request requires more-or-less a random seek + random rotation) (about 20ms?)*
    - \* *Even more detailed model might account for the fact that inodes are clustered together near outer edge of disk → reading lots of inodes may be a little bit faster (ls); reading first block of file may have a little longer seek*
  - *How many reads for first block of a random file?*
  - *How many reads to read a k-block file once the first block is read?*

- *How many writes to create a file?*
- *How many writes to write a k-block file?*
- *How long to do “ls” of a directory containing k files?*
- What is performance model for FFS? (what factors important)
  - 4KB blocks
  - sequential blocks of file layed out (mostly) sequentially (hopefully)
    - \* More detailed model accounts for skip-sector positioning, head switch time, cylinder switch time, optimization (p190) of limiting amount of file per cylendar group (first 48KB in first group; max of 1MB in each subsequent random group), ...
  - Inodes in cylendar groups
  - Files in same directory in same cylinder group; different directories in different groups
  - Paper “The biggest factor in this speedup is because of the larger block size” Do you agree?
- Specific questions/experiments/predictions?
  - How many reads for first block of a random file?
  - How long to do a read of first block of a random file?
  - How many reads to read a k-block file once the first block is read?
  - How long to read a k-block file once the first block is read?
  - How many writes to create a file?
  - How long to create a file?
  - How many writes to write a k-block file?
  - How long to write a k-block file?
  - How long to do “ls” of a directory containing k files?
  - ...
- Other notes
  - In paper: other factors (controller, CPU) limit performance. Not just disk model
    - \* Claim: Having good disk model would help you analyze system/write this paper by realizing that other factors were important.

## 7 Z

FS

### 7.1 K

Key features

- 128 bit (tech trend – 2x per year...1 bit per year)
- volume → pool
  - performance/correctness: understand transactions that span disks (v. low-level block interface must conservatively flush everything in order)
  - easier management (wrapper scripts can do some but not all e.g., grow a partition)
  - dynamic load balancing, grow pool, ...
- checksum data – unrecoverable read errors no longer can be ignored
  - Also proactively scrub
- copy on write – defer to LFS discussion

### 7.2 P

Performance v. ext3 journaled (slightly unfair to ZFS?)

v. ext3 ordered/writeback (very unfair to ZFS)

...probably won't make too much sense w/o LFS paper

assignment: after read LFS paper, see if graphs here make sense