Enterprise Storage

Low-Power Computing
Carnegie Mellon University
David Andersen

- Why do we need disks again?
  - Capacity
  - Persistence - *writes*
  - Redundancy / Backups - *writes*
  - Performance - IOps and throughput, read and *writes*
- DRAM caching can help reduce reads, but NVRAM is more $$$y. Starts to look like writes might be the hard part...
- Data point: Adaptec battery module: $150 to keep 128/256MB RAM active for 72 hours (scary? :)

Workload

- Going back to Osterhout & Douglis.
- Citing Amdahl's Law:
  - "CPU speeds are increasing dramatically ... but the speeds of disk drives are barely improving at all."
  - Large memories and disk arrays: "By 1995, we predict that disk arrays with 100s or even 1000s of disks will be standard products."
    - 2009: NetApp FAS6080: up to 1176 disks (!) 64GB memory

Log-Structured Filesystems

- Key to lots of good ways to use disks and flash (Rosenblum & Osterhout '92)
- Seeks are expensive, but disks have good sequential b/w.
- Ergo: Just write everything sequentially!
  - And then figure out how to find it / find latest version / etc.
  - Key: Write inode map periodically to end of log, keep it in memory for speed. Reads require seeks - just like always - but now writes are sequential.
- Practical consequence: Log cleaning - modified/deleted files still taking up space.

Log: Data & Metadata Layout

- Updated structures written to end of log
  - Buffer as long as possible to minimize seeks
  - Compare to FFS seek per 2 file inode writes, 1 file data write, 1 directory inode write, 1 directory data write (& free bitmap writes)

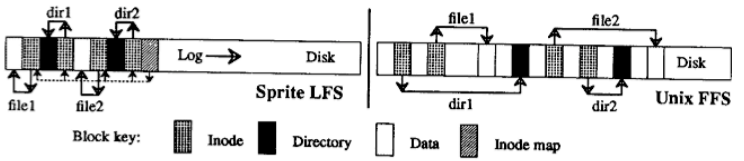


Fig. 1. A comparison between Sprite LFS and Unix FFS. This example shows the modified disk blocks written by Sprite LFS and Unix FFS when creating two single-block files named dir1/file1 and dir2/file2. Each system must write new data blocks and inodes for file1 and file2, plus new data blocks and inodes for the containing directories. Unix FFS requires ten nonsequential writes for the new information (the inodes for the new files are each written twice to ease recovery from crashes), while Sprite LFS performs the operations in a single large write. The same number of disk accesses will be required to read the files in the two systems. Sprite LFS also writes out new inode map blocks to record the new inode locations

Clever variants

- Journaling: Write to a small circular buffer (a log...) temporarily. Usually still flushes cache to disk except for failure recovery.
  - Many filesystems only journal metadata (crash can lose data, but can't corrupt entire filesystem)
- NetApp WAFL: Keep the log *anywhere* by writing data +inodes to any free block that the disk head is near. Update master pointers later (requires non-volatile storage of those pointers - usually in nvram)

Terminology

- from write-offloading paper
  - Volume: A single RAID-5 array (multiple disks used together). Writes are striped across all disks.
  - Active (spinning, in use); idle (spinning); standby (spun-down)

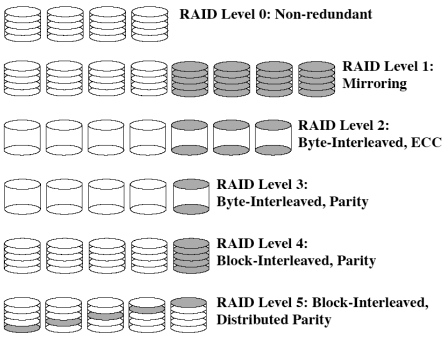
RAID

- Disks are slow
- Disks are unreliable
- Apply CS principle #3: Redundancy!

How reliable are they?

- Disks not as reliable as specs say
  - 3+% annual return rate
    - "Return" about as good as failure - if you yank the disk, you have to recover...
  - MTTF doesn't capture reality

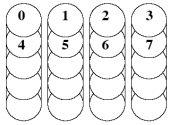
Synopsis of RAID levels



RAID levels

- RAID 0: Screamingly fast.
  - RAID 0 of 1000 drives: Screamingly *dead*...
- RAID 1: Mirroring
  - Really fast reads, *if* controller support
- RAID 2/3: Byte-interleaved (seems like bad idea)
  - Forced to access all disks for single read, even small
- RAID 4: Single parity disk
- RAID 5: Parity disk rotates
  - Difference not too huge in practice. Some major vendors use 4,

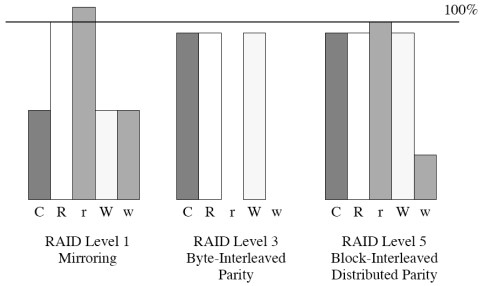
Striping: Read Throughput



- Goals:
  - Load balance high-concurrency, small accesses across disks
  - Enable parallel transfers for low-concurrency large reads
- Striping to the rescue
  - Uniform load for small reads
    - If striping unit contains the whole object (e.g., small read is contained on one disk)
  - Parallelism for large reads
    - Stripe unit small enough to spread read across many disks

Tradeoffs of RAID levels

- Relative to non-redundant, 16-disk array



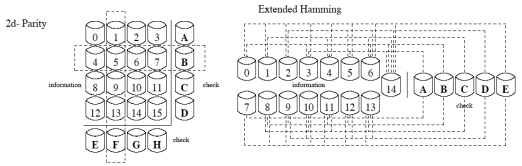
From "The Case for RAID"

Failures during Recovery

- Disks are now large -> Recovery time long
- Copying 1TB at 100MB/sec? > 3 hours
  In practice, much slower due to need to do RAID reconstruction
- Reconstruction stresses the disk
  ... which increases the failure rate.
- Consequence: Large arrays need more redundancy

Double Correcting

- Borrow from earlier approaches
  - Orthogonal parity groups (lec focus: 2D parity)
  - Double error-detecting codes from mem. systems



  - Overhead: check space vs. check update time
    - 2d parity: small time overhead (3), space = sqrt()

Write Offloading

- "Active" disks are up, handle writes
- Write traffic accounts, in their traces, for a lot of the "steady" workload; w/out writes, median active goes from 80% to 14%
- Note: Can handle reads for data that was "offloaded" -- but this doesn't help too much because those blocks are also likely to be in cache.

Sleep policy

- Threshold-based (seemed to work last time)
- Wake up when
  - Read to non-offloaded (and non-cached -- these traces are *after* FS buffer cache) block
  - # offloaded blocks > N

Eval

- Nice experimental eval point about open-loop vs closed-loop benchmarks -- they got queueing for later requests by running open-loop (which is worst-case for this load).
- Power savings: Decent
- 99.9th %ile response time: 800ms -> 15s
- 99th %ile: 100ms -> 200ms
- Writes remained happy.

Themes

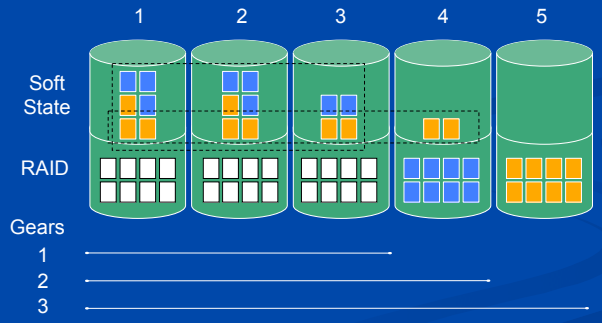
- Managing power in an aggregate / at a higher level can often expose more savings than managing components individually
  - Write offloading: "hey, we have other servers we can use"
  - PARAID: "hey, we have other disks..."
  - cf statistical multiplexing benefits in networking

More things we can trade

- Storage *space* for power
  - Why do we need disks again?
  - Capacity, ..., *performance*
  - (This isn't a new observation - space for performance is widely exploited)
- May change over time as storage systems age - initial provisioning, moving to "full"

Skewed Striping for Energy Saving

- Use over-provisioned spare storage
- Can use fewer disks for light loads



Data Layout

- Cascading parity updates
  - Must also update parity in soft state

d1 stores block 1, (1-4) (!!)

new parity block

	Disk 1	Disk 2	Disk 3	Disk 4	Disk 5
Soft State (RAID-5)	(1-4)	8	12	((1-4),8,12)	
	16	20	(16,20,_)	_	
RAID-5	1	2	3	4	(1-4)
	5	6	7	(5-8)	8
	9	10	(9-12)	11	12
	13	(13-16)	14	15	16
	(17-20)	17	18	19	20