Storage management: talk roadmap

▼ Why disk arrays?
   – Failures
   – Redundancy

▼ RAID

▼ Performance considerations
   – normal and degraded modes

▼ Disk array designs and implementations

▼ Case study: HP AutoRAID
Why disk arrays?

Because *stuff* happens.
Things break -- in a moderately predictable way in aggregate

Metrics:
- MTTF: “mean time to failure” -- a rate, not a period
- AFR: annual failure rate (better -- but still just middle of “bathtub”)
- MTTR: “mean time to repair”
Definitions

- **Reliability**
  - \( R(t) \) = likelihood system *up continuously* from time 0 to time \( t \)

- **Availability**
  - \( A(t) \) = likelihood system *will be up* at time \( t \)

- **Performability**
  - \( P(t,p) \) = likelihood system *will be providing performance* \( p \) *at time* \( t \)

![Diagram showing performance levels](image)
Solution: introduce redundancy!

▼ Complete copies
  – replication, “mirroring”

▼ Partial redundancy
  – Hamming codes/ECC
    • tolerates mangling of elements
    • unnecessarily strong: we know when disks are broken
  – Parity
    • XOR sets (stripes) of data blocks to calculate a “parity block”
    • any data block in a stripe can be reconstructed from the others + parity
Redundancy helps

▼ For disks:
  – originally (mid-1980s), these were the most unreliable components
  – nowadays, they’re one of the more reliable ones (AFR of 1-2%)
  – but failure rates are proportional to numbers …

▼ Assume: independent failures
  warning! danger! caution! error!

▼ With no redundancy …
  \[ \text{AFR}_{\text{disks}} \approx N_{\text{disks}} \times \text{AFR}_{\text{disk}} \]

▼ With one degree of redundancy …
  \[ \text{AFR}_{\text{raid}} \approx \text{AFR}_{\text{disks}}(N_{\text{disks}}) \times \text{MTTR}_{\text{disk}} \times \text{AFR}_{\text{disks}}(N_{\text{disks}}-1) \]
Redundancy hurts, too

▼ Cost
- replicating everything costs 2x as much storage
- solution: partial redundancy

▼ Slower updates
- 2x as many copies to write to
- ... even worse with partial redundancy

▼ Greater complexity
- 80-90% of disk array firmware is error handling
- lots and lots of configuration choices ...
Storage management: talk roadmap

▼ Why disk arrays?
  – Failures
  – Redundancy

▼ RAID

▼ Performance considerations
  – normal and degraded modes

▼ Disk array designs and implementations

▼ Case study: HP AutoRAID
Originally (like everything else?) invented by IBM
- striping explored at Univ. Maryland

- catchy terminology popularized by UC Berkeley:
  - Patterson, Gibson & Katz: “The case for Redundant Arrays of Inexpensive Disks (RAID)”, *ACM SIGMOD, 1988*

- comparison point: slow, large expensive disks (SLED)
  - goal was to compete with IBM mainframe disks using cheap, unreliable PC drives

Now:

**RAID: Redundant Arrays of Independent Disks**
RAID levels 0, 1, 10

▼ RAID0: striping (no redundancy)

**Striping** balances the load and allows large transfers to happen in parallel

▼ RAID1: aka mirroring (full redundancy)

**Mirroring** gives 2x the read bandwidth per disk, but writes have to go to both

▼ RAID10: striped mirroring (full redundancy)
RAID level 3 - parity-protected striping

- RAID3: byte-interleaved
  - all disks read/written in lock step
  - parity on dedicated disk
    (ok: sees same load as remainder)
  - great for high-bandwidth, large transfers; otherwise poor

**XOR parity** is single-bit ECC that can correct single-bit erasures

If a disk is missing, XORing the others will give you its contents

Parity unit (xor of rest of stripe units in same stripe)
RAID level 4 - block-interleaved stripes

- **RAID4**: block-interleaved
  - independent reads/writes possible
  - parity on dedicated disk (hot spot!)

- Updating parity is expensive for small writes
  - one effect: write-caching becomes especially important

1. Read old data & parity
2. Compute new parity
3. Write new data + parity

=> 4x I/O operations per small write
RAID level 5

- RAID5 - rotated-parity-protected striping to balance the load

Parity unit (xor of rest of stripe units in same stripe)

Rotating the parity balances the parity load across all the disks; striping allows fast large transfers

RAID5 is the configuration of choice for all but performance-intensive loads
Currently accepted RAID levels

- 0: no redundancy
- 1: full copy (mirrors)
- 10: striped mirrors

- 2: Hamming-code/ECC (not used)
- 3: byte-interleaved parity
- 4: block-interleaved parity (more useful variant of RAID3)
- 5: rotated block-interleaved parity
- 6: double parity (“P+Q parity” -- rare)

*Note: not really levels, just a list*
RAID: some tricky points

▼ Updates in flight at time of power failure can corrupt the parity
  – either: an expensive parity rebuild on power up
  – or: keep a non-volatile intentions log

▼ Reliability calculations based on disks alone are bogus
  – power source is single largest problem
  – then controller failures, cooling, backplane, connections, ...
  – redundancy helps here, too
  – nobody likes to talk about software …
Storage management: talk roadmap

▼ Why disk arrays?
  – Failures
  – Redundancy

▼ RAID

▼ Performance considerations
  – normal and degraded modes

▼ Disk array designs and implementations

▼ Case study: HP AutoRAID
Floating parity [Menon92]
- write parity anywhere -- saves one revolution

1. Read old data & parity
   * claim: old data is already cached
2. Compute new parity
3. Write new data + parity
   * parity write is “free”
=> ~2x I/O operations per small write
Parity logging [Stodolsky94]
- aggregate parity updates into an append-only log
- propagate log in background
Improving performance: AFRAID

- A Frequently Redundant Array of Independent Disks [Savage&Wilkes96]
  - live (a little) dangerously ...
  - update parity opportunistically in the background
  - gives smooth tradeoff between availability and performance
Improving performance: choice of stripe size

- Optimum size is dependent on:
  - read:write mix
  - data layout
    (RAID1 vs RAID5)
  - concurrency level
  - back-end disk characteristics
    (e.g., track size)

- Choices are a daunting problem for sysadmins

[Chen90] for RAID1, [Chen95] for RAID5
Degraded-mode performance

- Reading RAID4/5 when a disk is broken is expensive

1. Read all surviving data & parity
2. Compute missing data (XOR)

=> all surviving disks are involved
Improving RAID5 degraded mode performance

- Chained declustering [Hsaio+DeWitt90]
  - spread the second copy out over other disks
  - when the primary copy breaks, each secondary disk takes up only a portion of the slack

![Diagram showing chained declustering]

Primary copy

Secondary copy, spread over other drives
Improving RAID5 degraded mode performance

Declustering [Muntz90]
- make stripes narrower than whole array
  - only stripes that have the broken disk need pay performance penalty
- each stripe uses a different set of disks
  - some complexity in the mappings that do this nicely
  - but “close enough” works just fine
- better degraded-mode performance, at the cost of more disks
  - stripes are smaller => more parity
- improvements
  - Approximate block designs
  - Prime/Relpr [Alvarez98] - better-spread large-transfer load
Recovery/rebuild after a disk failure

- Reduce MTTR: keep an online spare
  - e.g., XP256: up to 4 spares per rack of 64 drives

- Distributed sparing [Menon92] makes the spare useful
  - spread its “contents” across all the disks
  - effectively adds an extra disk’s performance to the array
Recovery/rebuild after a disk failure

Reconstruction after failure

- sweep across data: read every stripe, rewrite parity/missing data
  - poor performance if done too simply: data transfers are too small; too much blocking
  - better: disk-oriented reconstruction [Holland93]
    - keep $\geq$ one outstanding read for each disk
- can also piggyback updates on foreground activity
  - requires keeping a map of reconstructed stripes

- big tradeoff: faster recovery or slower foreground activity?
Storage management: talk roadmap

▼ Why disk arrays?
   – Failures
   – Redundancy

▼ RAID

▼ Performance considerations
   – normal and degraded modes

▼ Disk array designs and implementations

▼ Case study: HP AutoRAID
Array implementations

▼ In software
  – cheapest, but consumes memory (and cpu) cycles
  – usually mirroring, in OS Logical Volume Manager

▼ In host bus adapter
  – common in PC servers
  – big win is from the read/write cache
  – fault handling is very limited
Array implementations

- **Mid-range array** (e.g., HP FC60)
  - sometimes separate controller and disk boxes
  - up to 1-2TB disk, 0.5Gb cache RAM
  - can saturate a 100MB/s FibreChannel link; $O(10,000 \text{ IOs/s})$

Packaging:
- whole array is in a single box, or
- array controller is in separate box
Array implementations

▼ High-end array: integrated box (e.g., HP XP256; EMC Symmetrix)
   – up to a few TB of disk
   – up to a few GB of cache
   – up to a few $million

▼ What you pay for:
   – lots of caching (vital to performance)
   – multiple host interface types
     • e.g., HP XP256: SCSI, FibreChannel, and ESCON
   – quality power distribution, cabling, cooling, vibration isolation
   – phone-home, remote management, support infrastructure
   – can saturate a few 100MB/s FibreChannel links; O(50,000+ IOs/s)
Disk array architecture - high end

Packaging:
- array controller in a separate box (disks in rack-mountable trays), to
- array controller is one of several 6’ racks

Diagram:
- Front-end controllers
- Host interfaces
- Cache
- Power, etc
- Parity XOR
- Back-end controllers
Most arrays provide multiple LUNs (SCSI Logical UNIts)
- one or more disk drives bound together into a common layout
- different LUNs can have different sizes, different layouts
- LUN 0 is often special (used for controlling the array as a whole)
- at low end: 8-32 LUNs
- at high end: thousands of LUNs
  - SCSI limit: 4096 LUNs, from a 12 bit LUNid

A few common variations (there are many more):
- parts of disks instead of whole disks
- LUNs may be named relative to ports, not uniquely
- LUNs can have different caching behavior
Summary

- Disk arrays use redundancy to protect against disk (and other storage component) failures
  - rest easy: the storage system is no longer the main problem!

- They can also provide performance benefits
  - caching can easily provide 10-100x performance boost

- But … at cost of lots of complexity
  - algorithms
  - configuration choices
  - implementation options
Storage management: talk roadmap

- Why disk arrays?
  - Failures
  - Redundancy

- RAID

- Performance considerations
  - normal and degraded modes

- Disk array designs and implementations

- Case study: HP AutoRAID