Storage Systems

NPTEL Course

Jan 2012

(Lecture 26)

K. Gopinath

Indian Institute of Science

Types of Disk Redundancy

- Maximum Distance Separable (MDS) vs non-MDS
- MDS: RAID1, RAID4, RAID5, RAID6 popular
  - RAID1: mirroring
  - RAID5: parity rotated but not in RAID4
  - RAID m+n where RAID m used as leaves and RAID n on top
    - RAID10: create multiple RAID1 (mirroring) volumes and then catenate them in RAID 0
  - RAID6: need two syndromes for tolerating 2 failures
    - 1st one the regular RAID5: xor of $D_0,...D_i,...,D_{n-1}$
    - 2nd one: xor of $g^0D_0,...,g^iD_i,...,g^{n-1}D_{n-1}$, g a generator in a GF
    - If one disk fails, RAID5 syndrome sufficient
    - If 2 data disks fail, say ith and jth: have to solve $D_i+D_j=A$ and $g^iD_i + g^jD_j=B$

RAID5 “write-hole”

- Software RAID5 perf poor
  - if data upd in a RAID stripe, must also upd parity
    - If data part upd but crash/power outage before parity upd, xor invariant of RAID stripes lost
  - a full-stripe write can issue all writes asynch, but a partial-stripe write must do synch reads before it can start the writes
- Also, a partial-stripe write modifies live data
  - defeats transactional design
- software-only workarounds: logging but slow!
- HW workaround:
  - NVRAM for both probs but costly...

RAID-Z

- Dynamic striping across all devices to maximize throughput as additional devices added to zpool
- On read of a RAID-Z block, ZFS compares it against its checksum. If no match, ZFS reads parity and does combinatorial reconstruction to figure out which disk returned bad data
- Write to a new location and then atomically overwrite the pointer to the old data
  - Uses COW: no overwriting data
- Variable stripe size so that every write a non-RMW
  - eg. mirroring for small writes instead of parity protected

- On RAID-Z reconstruction, traverse filesystem metadata to determine the RAID-Z geometry
    - Tradeoff: expensive when pool full
    - Not possible in designs that separate FS and VM
    - Bonus: traversing metadata => ZFS can validate every block against its 256-bit checksum (std RAID can only XOR without any check)
- ZFS RAID differs from std RAID solutions
  - Reconstructs only live data and metadata when replacing a disk
    - not the entirety of disk including blank and garbage blocks
  - replacing a partially full disk in a ZFS pool takes proportionately less time compared to std RAID
- No special hardware (eg. no NVRAM for correctness, no write buffering for good performance)

NetApp RAID4

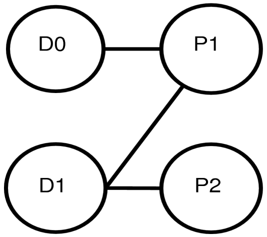
- If files are scattered on disk and fs does not understand layout, every write may require costly seeks for parity writes
  - Rotating parity in RAID5 may help as it has multiple disk arms for rotated parity
- Similar to ZFS, WAFL (Write Anywhere File Layout) understands RAID4 layout
  - Can write data and metadata close together as it does not do in-place upd
  - Parity disk may need to do no or small seeks only

Types of Disk Redundancy

- Maximum Distance Separable (MDS) vs non-MDS
- MDS: RAID1, RAID4, RAID5, RAID6 popular
  - RAID1: mirroring
  - RAID5: parity rotated but not in RAID4
  - RAID m+n where RAID m used as leaves and RAID n on top
    - RAID10: create multiple RAID1 (mirroring) volumes and then catenate them in RAID 0
  - RAID6: need two syndromes for tolerating 2 failures
    - 1st one the regular RAID5: xor of $D_0, \ldots D_i, \ldots, D_{n-1}$
    - 2nd one: xor of $g^0 D_0, \ldots, g^i D_i, \ldots, g^{n-1} D_{n-1}$, g a generator in a GF
    - If one disk fails, RAID5 syndrome sufficient
    - If 2 data disks fail, say i^{th} and j^{th}: have to solve $D_i + D_j = A$ and $g^i D_i + g^j D_j = B$

Non-MDS based Redundancy

- Non-MDS codes have varying capability to recover from multiple errors but simpler as no solutions involving GF required



  - consider a linear (9,6) array with 9 data disks and 6 parity disks.
  - the best possible XOR-coding has a Erasures Vector (EV) of [0,0,0,30,390,2230]
  - can tolerate up to 3 disk failures without data loss (the first 3 entries); however, there is data loss in
    30 configurations with 4 disk failures,
    390 configurations with 5 disk failures and
    2230 configurations with 6 disk failures.

Flash and Storage Systems

- Flash: based on semiconductor tech
  - 19nm, 8GB flash chip with 2 bits per cell; lower power?
  - Erase, read and write sizes
    - COW at device level
  - Wear endurance: newer reliability models?
    - Dynamic/static wear levellling, garbage collection
    - Write amplification
  - raw and SSD

- Enterprise Flash becoming common
  - IOP/disk is approx 50/sec
  - IOP on flash:
    - 2-8k IOPs per SSD currently (8k random R/W)
    - One order higher with PCIe Flash
      - eg. ratings of 90,000 IOPS / 38,000 IOPS on random 4K R/W

FS on flash?

- TRIM command for SSDs
  - When block dealloc, flash layer needs to be told
- FS on raw flash?
- Manage wear, parallelism (Flash Translation Layer)
  - Write buffer to absorb random writes
- Linux: mtd device: 512B block device but RMW!
    - mtdblock driver buffers writes and writes to flash when full
  - NFTL: Linked list of replacement blocks on rewrites
    - GC “folds” longest chains
  - DFTL: uses SRAM to store recent maps
- Natural fit with LFS?
  - YAFFS2 on mtdblock device: a LFS
  - 3 levels of GC: background, passive and aggressive

Parallelism

- SSD: 1+ channels.
  - Channel: 1+ flash packages.
  - Flash package: 1+ flash chips
  - Flash chip: 1+ dies
  - 1 die: 1+ flash planes
  - Raw flash can expose most of the parallelism (64x?)
- Std Linux I/O Q'ing not suitable
  - Need to develop new models
  - Diff in R/W sizes, Interrupt overhead: polling?
  - Scheduling R/W/Erase, GC, prefetch
- Higher IOPs: newer bottlenecks in system?

Summary

- FS design complex
  - Multiple types of clients
  - Have to bridge mem and disk speeds
    - High levels of concurrency needed
    - Newer types of tech (flash, SCM)
      - SCM: as memory? New security issues?
  - Serious layering issues
- Any error in fs code can result in loss of data
  - Have to contend with hw errors also!
  - Integration of volume manager with fs
  - HW RAID vs SW RAID
  - Or, drop RAID support in kernel code/HW and do it across independent systems using standard networking code at user level (as in GFS)