CS 61C

Discussion 12: Dependability (ECC, RAID, I/O)

John Yang
Summer 2019

Announcements

- Homeworks 7, 8 due today
- Check In: tinyurl.com/john61c
- Final on Thursday!

Today's Goal

- Machines break, systems fail, and in general, things can go wrong in the real world. Big Question: How can we design or coordinate systems such that reliability a.k.a. dependability is guaranteed?
- General Theme: Ascertain dependability through redundancy + replication!

Dependability

Define and quantify what it means for a service to be dependable

Terminology

Dependability: Measure of system's...

- Reliability: Ability of system to function under stated conditions for period of time
  - Reliability = Mean Time To Failure (MTTF)
- Availability: Proportion of time a system is in a functioning, usable condition
  - Mean Time To Repair (MTTR), Mean Time Between Failures (MTBF) = MTTR + MTTF
  - Availability = MTTF / MTBF, to improve availability
    - Increase MTTF: More reliable hardware + software, fault tolerant systems
    - Decrease MTTR: Improved tools + processes for diagnosis and repair
- Maintainability [Out of Scope]: Ease with which service's errors, defects can be repaired

Fault: Failure of any component of a system, may or may not cause system failure

- "Fault Tolerance": Property implying a system can continue operating properly despite the failure of a number of components.

Miscellaneous

Reliability

- MTTF, MTBF quantified as hours per failure
- Annualized Failure Rate (AFR): Probability a component will fail after year of use
  - Generally, relates MTBF and hours a number of devices are run
  - I.e., for HDD: (Disks / MTTF x 8760 hr/yr) x 1 / Disks = 8760 hr / yr / MTTF

Availability

- Usually written as percentage, MTTF / (MTTF + MTTR)
- Number of “nines” ⇒ 1 “9” = 90%, 2 “9” = 99% …

ECC

Error Correction Code

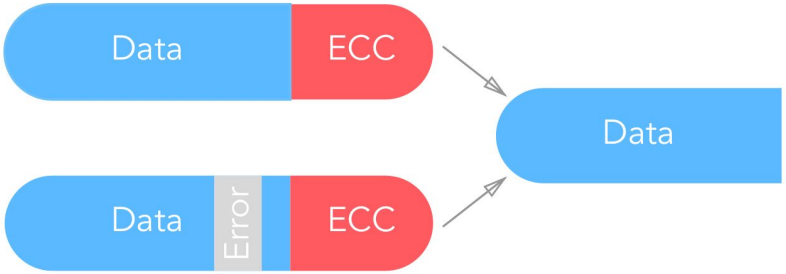
Error Correction Codes

Motivation: Memory systems are prone to errors. A common issue is bits accidentally being flipped. Traditional DRAMs store very little charge per bit.

Solution: Detect errors by storing outputs from a *function* on the data in extra bits.

- Detection: function(data) fails to match bits' values, data is "invalid"
- Correction: Invalid data can be mapped to a satisfactory, valid form

Visualization



Functions on data ⇒ Store in Extra Bits

Hamming Distance: Count number of bits that are different

Parity: Count number of 1 Bits

Parity Bit

Illustrated

Parity Bit: Add extra bits so data must be even parity.

We aren't limited to a single bit. Detection doesn't *have* to be binary!

- Multiple bits ⇒ Afford larger modularities
  i.e. 3 "Parity Bits" ⇒ Modulo 8! (2^3)
- Decreases likelihood that parity bits fail to catch error due to multiple errors
  - 1 Parity Bit: Fails to catch error if # of errors is multiple of 2. (½ Correct Parities)
  - 3 Parity Bits: Fails to catch error if # of errors is multiple of 8. (⅛ Correct Parities)

Hamming ECC

Illustrated

Hamming ECC: Insert parity bits at each each position that is a power of 2.

- Place parity bits between data to form code word.
- Other bit positions are for data bits (3, 5, 6, 7...)

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	✗		✗		✗		✗		✗		✗		✗		✗		✗		✗		...
	p2		✗	✗			✗	✗			✗	✗			✗	✗			✗	✗		
	p4				✗	✗	✗	✗					✗	✗	✗	✗					✗	
	p8								✗	✗	✗	✗	✗	✗	✗	✗						
	p16																✗	✗	✗	✗	✗	

Hamming ECC

Illustrated

Example:

Given a set of bits such as 101011010, how do we construct the Hamming ECC bits?

1. All bit positions in code word at powers of 2 are for parity bits (1st Bit = Index 1)

_, _, 1, _, 0, 1, 0, _, 1, 1, 0, 1, 0

2. Set each parity bit such that the parity of its group of bits (i.e. Parity Bit 1, the leftmost bit, has a group of bits 1, 3, 5, 7, 9, 11...) is even

- Parity bit's *position* determines group of bits it checks (is position number's bit value = 1)

Bit 1 (0001_2) checks bits: 1 ($000\underline{1}_2$), 3 ($00\underline{1}1_2$), 5 ($0\underline{1}01_2$), 7 ($0\underline{1}11_2$), 9 ($1\underline{0}01_2$)... Pattern: ($XXX1_2$)
Bit 2 (0010_2) checks bits: 2 ($00\underline{1}0_2$), 3 ($00\underline{1}1_2$), 6 ($0\underline{1}10_2$), 7 ($0\underline{1}11_2$), 10 ($1\underline{0}10_2$)... Pattern: ($XX1X_2$)
Bit 4 (0100_2) checks bits: 4 ($0\underline{1}00_2$), 5 ($0\underline{1}01_2$), 6 ($0\underline{1}10_2$), 7 ($0\underline{1}11_2$), 12 ($1\underline{1}00_2$)... Pattern: ($X1XX_2$)

Hamming ECC

Illustrated

2. (Continued)

Bit 1: 0b_, _, 1, _, 0, 1, 0, _, 1, 1, 0, 1, 0 ⇒ Parity Bit is 0

Bit 2: 0b_, _, 1, _, 0, 1, 0, _, 1, 1, 0, 1, 0 ⇒ Parity Bit is 1

Bit 4: 0b_, _, 1, _, 0, 1, 0, _, 1, 1, 0, 1, 0 ⇒ Parity Bit is 0

Bit 8: 0b_, _, 1, _, 0, 1, 0, _, 1, 1, 0, 1, 0 ⇒ Parity Bit is 1

Result: 0b01100101111010, red highlight denotes hamming bits

Hamming ECC

Illustrated

The Hamming ECC bits not only help us *detect* errors, but can also *correct*, too.

Example

Given 0b01100101111010 from before, let's flip a random bit ⇒ 0b01100101111110

Detection

P1: 0b 0 1 1 0 0 1 0 1 1 1 1 1 0 ⇒ Odd! Error
P2: 0b 0 1 1 0 0 1 0 1 1 1 1 1 0 ⇒ Odd! Error
P4: 0b 0 1 1 0 0 1 0 1 1 1 1 1 0 ⇒ Even. No Error
P8: 0b 0 1 1 0 0 1 0 1 1 1 1 1 0 ⇒ Odd! Error

Correction

P1, P2, and P8 Error-ed. By referring to the chart's columns on Slide 9, we can determine that the 7th Bit of the original data or 11th Bit of the Hamming sequence is erroneous!

RAID

Redundant Array of Independent Disks

RAID

Motivation: Disks can fail and lose information when they do! How can we improve availability and reliability of storage in this context?

Solution: Redundancy! Combine multiple disk drive components into a single logical unit. File data is striped across multiple disks. If a single disk fails...

- The data is still intact on another disk ⇒ high availability
- Data can be reconstructed from redundant storage on other disks

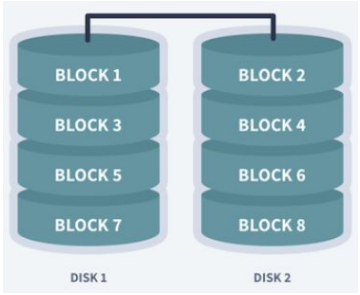
Fast Facts

- Created here at UC Berkeley by David Patterson, Randy Katz, Garth A. Gibson
- At publication, beat out contemporary mainframe disk drives' performance
- Research paper linked here

Data Replication Techniques

Striping: Divide a body of data into blocks, spread data blocks across multiple storage devices (i.e. Hard Disks, SSDs) for performance improvement

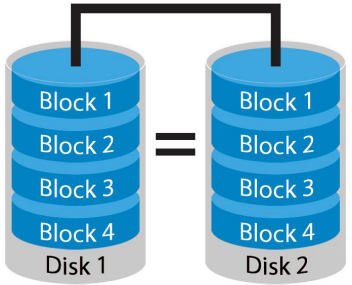
- Multiple disks ⇒ faster reads, writes
- Stripe can be at different amounts (byte, block, partition)
- *Not* fault tolerant; Failed disk ⇒ irrecoverably lost data



Data Striping

Mirroring: Store exhaustive copies of data on 2+ disks

- Improved Read Speed + Fault Tolerant to disk failures
- Slower Writes (repeated writes to multiple disks)
- Extremely resource intensive (a lot of storage required!)



Data Mirroring

Parity: Single data bit indicating whether message's number of bits is odd or even, provides fault tolerance.

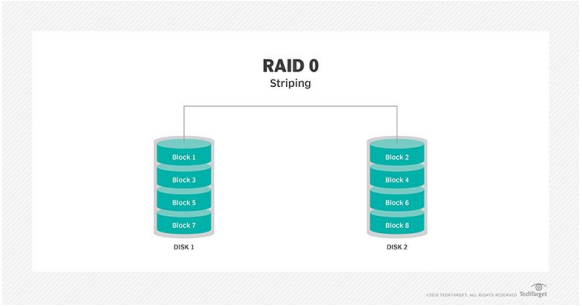
RAID Versions

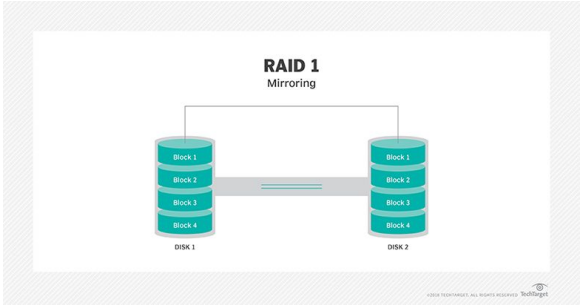
RAID	0	1	2-4	5
Features	Striping	Mirroring	Striping (Bit/Byte/Block) + Parity Disk (contains other disks' parity)	Striping (Block) + Distributed Parity (across 2+ disk drives)
Pros	Concurrent Read + Write, High Throughput	High Availability + Read Performance	High availability, less storage required than RAID 1 (parity disk)	Both Data + Parity striped evenly ⇒ no bottleneck
Cons	Any Disk Failure ⇒ Every file is lost.	Slow writes (must update all drives), resource intensive, requires min 2 disks	Slow writes (must update Parity Data)	Requires min 3 disks, data recovery / rebuild takes long time
Modern Use?	No redundancy	Too expensive	2: Obsolete 3: Bad I/O Rates	5: Considered most secure RAID config.

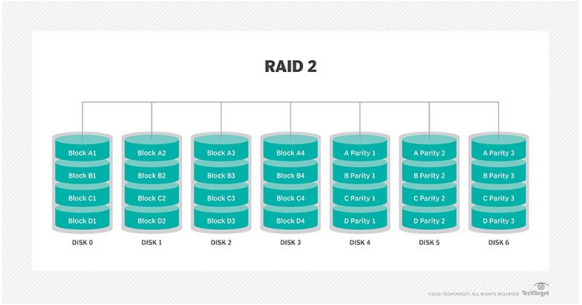
RAID Summary

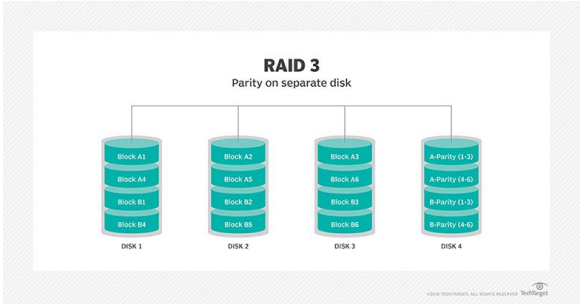
Traditional RAID levels

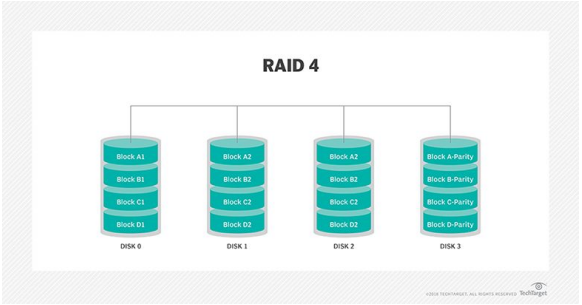
LEVEL	DESCRIPTION
RAID 0	■ Data striped across hard disk drives for maximum write performance ■ No actual data protection
RAID 1	■ Synchronously mirrors all data from each HDD to an exact duplicate HDD ■ No data lost if HDD faults or fails ■ Typically highest-performing RAID level at the expense of lower usable capacity
RAID 2	■ Data protected by error correcting codes ■ Parity HDD requirements proportional to the log of HDD number ■ Somewhat inflexible and less efficient than RAID 5 or RAID 6 with lower performance and reliability ■ Not widely used
RAID 3	■ Data is protected against the failure of any HDD in a group of N+ ■ Similar to RAID 5, but blocks are spread across HDDs ■ Parity is bitwise vs. RAID 5 block ■ Parity resides on a single HDD rather than being distributed among all disks ■ Random write performance is quite poor, and random read performance fair at best
RAID 4	■ Similar to RAID 3, stripes data across many HDDs in blocks instead of RAID 3 bytes to improve random access performance ■ Data protection is provided by a dedicated parity HDD ■ Similar to RAID 5 except uses dedicated parity instead of distributed parity ■ Dedicated parity HDD remains a bottleneck, especially for random write performance
RAID 5	■ Most common RAID ■ Provides RAID 0 performance with more economical redundancy ■ Stripes block data across several HDDs while distributing parity among the HDDs ■ Uses HDDs more efficiently, providing overlapped read/write operations ■ Provides more usable storage than RAID 1 or RAID 10 ■ Data protection comes from parity information used to reconstruct data of a failed drive ■ Minimum of three and usually five HDDs per RAID group ■ Rebuilds cause lower storage system performance ■ Potential total RAID group data loss if second drive fails during rebuild ■ Read performance tends to be lower than other RAID types because parity data is distributed on each HDD
RAID 6	■ Similar to RAID 5, but includes a second parity scheme distributed across the HDDs of the RAID group ■ Dual parity protects against data loss if second HDD fails ■ Tends to have lower storage system performance than RAID 5 and can plummet during dual HDD rebuilds
RAID 10 (RAID 1 + RAID 0)	■ RAID 1 striped ■ Improves write performance
RAID 50 (RAID 5 + RAID 0)	■ RAID 5 striped ■ Improves write performance closer to RAID 1
RAID 60 (RAID 6 + RAID 0)	■ RAID 6 striped ■ Improves write performance closer to RAID 1

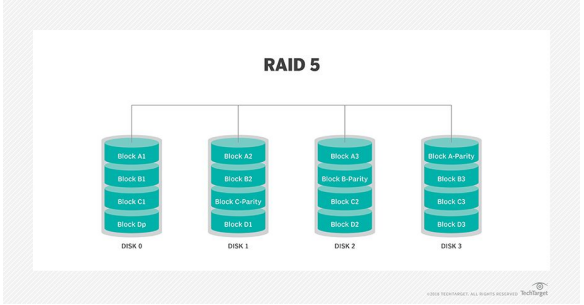












I/O

Input / Output: Communication between a computer and the outside world.

I/O

Motivation: So far, we've covered memory, caches, CPU, and disk. However, how does our computer interface with other devices?

Solution: Input / Output is the process of interacting with both internal (graphics cards) and external (mouse, keyboard) devices connected to the computer.

I/O Types

- Polling: Constantly check *control register* for new data, read if available.
  - Preferable for low-latency devices that continually have new data (i.e. Mouse)
- Interrupt: Device sends *interrupt* signal to OS when ready. Interrupt stops running code, handles device data, then resumes execution.
  - Each Interrupt = Performance Hit, this approach is better for devices with fewer requests

I/O Hardware

Registers

- Control: Indicates whether it’s ok to read from or write to a device
- Data: Register that holds data being read + serves as write destination
- The status + values of both registers are controlled by the device

Device Driver

- Program that operates + controls device connected to computer
- Acts as software abstraction + interface for hardware, allows OS to access hardware’s functionality. Translates hardware actions ⇔ OS commands.

Memory Mapped I/O

Motivation: Polling + Interrupts take up CPU cycles, does the CPU have to be responsible for I/O?

Idea: Devices themselves handle data transfer.

- CPU defines spots in the address space that devices can write to directly.
- System interacts with device through load / store instructions on device's address range (eliminates CPU's participation

Benefits:

- No more polling, interrupts, or CPU computation cycles are required!
- Memory mapped I/O uses *same address space* for memory + I/O devices ⇒ CPU can use same technique to access physical RAM or I/O device's memory