15-440 Distributed Systems

Lecture 11 – Errors and Failures

Types of Errors

- **Hard errors**: The component is dead.
- **Soft errors**: A signal or bit is wrong, but it doesn't mean the component must be faulty
- Note: You can have recurring soft errors due to faulty, but not dead, hardware

Examples

- DRAM errors
  - Hard errors: Often caused by motherboard - faulty traces, bad solder, etc.
  - Soft errors: Often caused by cosmic radiation or alpha particles (from the chip material itself) hitting memory cell, changing value. (Remember that DRAM is just little capacitors to store charge... if you hit it with radiation, you can add charge to it.)

Some fun #s

- Both Microsoft and Google have recently started to identify DRAM errors as an increasing contributor to failures... Google in their datacenters, Microsoft on your desktops.
- We've known hard drives fail for years, of course. :)

Replacement Rates

HPC1		COM1		COM2	
Component	**%**	**Component**	**%**	**Component**	**%**
Hard drive	30.6	Power supply	34.8	Hard drive	49.1
Memory	28.5	Memory	20.1	Motherboard	23.4
Misc/Unk	14.4	Hard drive	18.1	Power supply	10.1
CPU	12.4	Case	11.4	RAID card	4.1
motherboard	4.9	Fan	8	Memory	3.4
Controller	2.9	CPU	2	SCSI cable	2.2
QSW	1.7	SCSI Board	0.6	Fan	2.2
Power supply	1.6	NIC Card	1.2	CPU	2.2
MLB	1	LV Pwr Board	0.6	CD-ROM	0.6
SCSI BP	0.3	CPU heatsink	0.6	Raid Controller	0.6

Measuring Availability

- Mean time to failure (MTTF)
- Mean time to repair (MTTR)
- MTBF = MTTF + MTTR
- Availability = MTTF / (MTTF + MTTR)
  - Suppose OS crashes once per month, takes 10min to reboot.
  - MTTF = 720 hours = 43,200 minutes
    MTTR = 10 minutes
  - Availability = 43200 / 43210 = 0.997 (~"3 nines")

Availability

Availability %	Downtime per year	Downtime per month*	Downtime per week
90% ("one nine")	36.5 days	72 hours	16.8 hours
95%	18.25 days	36 hours	8.4 hours
97%	10.96 days	21.6 hours	5.04 hours
98%	7.30 days	14.4 hours	3.36 hours
99% ("two nines")	3.65 days	7.20 hours	1.68 hours
99.50%	1.83 days	3.60 hours	50.4 minutes
99.80%	17.52 hours	86.23 minutes	20.16 minutes
99.9% ("three nines")	8.76 hours	43.8 minutes	10.1 minutes
99.95%	4.38 hours	21.56 minutes	5.04 minutes
99.99% ("four nines")	52.56 minutes	4.32 minutes	1.01 minutes
99.999% ("five nines")	5.26 minutes	25.9 seconds	6.05 seconds
99.9999% ("six nines")	31.5 seconds	2.59 seconds	0.605 seconds
99.99999% ("seven nines")	3.15 seconds	0.259 seconds	0.0605 seconds

Availability in practice

- Carrier airlines (2002 FAA fact book)
  - 41 accidents, 6.7M departures
  - 99.9993% availability
- 911 Phone service (1993 NRIC report)
  - 29 minutes per line per year
  - 99.994%
- Standard phone service (various sources)
  - 53+ minutes per line per year
  - 99.99+%
- End-to-end Internet Availability
  - 95% - 99.6%

Real Devices

Real Devices – the small print

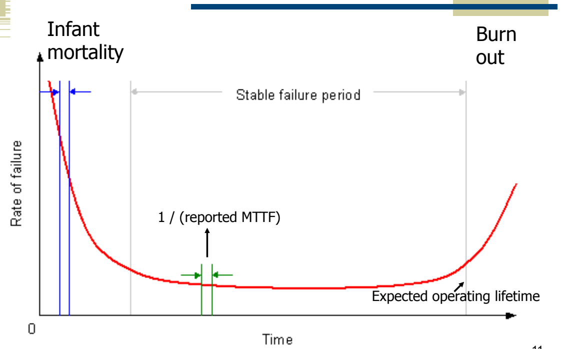
- The Cheetah™ 15K.4 is the highest-performance drive ever offered by Seagate®, delivering maximum IOPS with fewer drives to yield lower TCO.
- The Cheetah 15K.4 price-per-performance value united with the breakthrough benefits of serial attached SCSI (SAS) make it the optimal 3.5-inch drive for rock solid enterprise storage.
- Proactive, self-initiated background management functions improve media integrity, increase drive efficiency, reduce incidence of integration failures and improve field reliability.
- The Cheetah 15K.4 shares its electronics architecture and firmware base with Cheetah 10K.7 and Savvio™ to ensure greater factory consistency and reduced time to market.

KEY SPECIFICATIONS

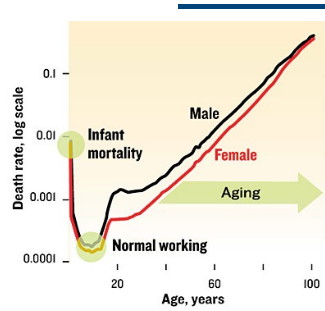
- 146-, 73- and 36-Gbyte capacities
- 3.3-msec average read and 3.8-msec average write seek times
- Up to 96 Mbytes/sec sustained transfer rate
- 1.4 million hours full duty cycle MTBF
- Serial Attached SCSI (SAS), Ultra320 SCSI and 2 Gbits/sec Fibre Channel interfaces
- 5-year warranty

For more information on why 15K is the industry's best price/performance disc drive for use in mainstream storage applications, visit **http://specials.seagate.com/15k**

Disk failure conditional probability distribution - Bathtub curve



Other Bathtub Curves



Human Mortality Rates (US, 1999)

From: L. Gavrilov & N. Gavrilova, "Why We Fall Apart," IEEE Spectrum, Sep. 2004. Data from http://www.mortality.org

So, back to disks...

- How can disks fail?
  - Whole disk failure (power supply, electronics, motor, etc.)
  - Sector errors - soft or hard
    - Read or write to the wrong place (e.g., disk is bumped during operation)
    - Can fail to read or write if head is too high, coating on disk bad, etc.
    - Disk head can hit the disk and scratch it.

13

Coping with failures...

- A failure
  - Let's say one bit in your DRAM fails.
- Propagates
  - Assume it flips a bit in a memory address the kernel is writing to. That causes a big memory error elsewhere, or a kernel panic.
  - Your program is running one of a dozen storage servers for your distributed filesystem.
  - A client can't read from the DFS, so it hangs.
  - A professor can't check out a copy of your 15-440 assignment, so he gives you an F.

14

Recovery Techniques

- We've already seen some: e.g., retransmissions in TCP and in your RPC system
- Modularity can help in failure isolation: preventing an error in one component from spreading.
  - Analogy: The firewall in your car keeps an engine fire from affecting passengers
- Today: Redundancy and Retries
  - Two lectures from now: Specific techniques used in file systems, disks
  - This time: Understand how to quantify reliability
  - Understand basic techniques of replication and fault masking

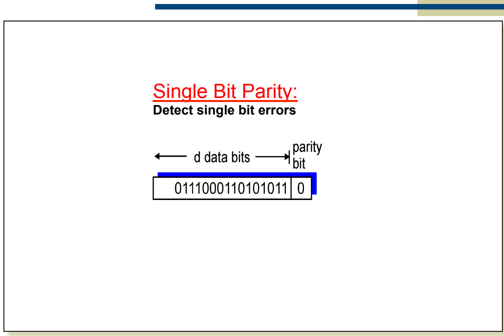
15

What are our options?

1. Silently return the wrong answer.
2. Detect failure.
3. Correct / mask the failure

16

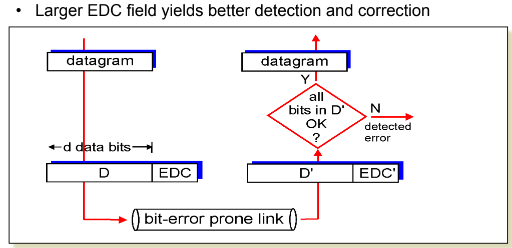
Parity Checking



17

Block Error Detection

- EDC= Error Detection and Correction bits (redundancy)
- D = Data protected by error checking, may include header fields
- Error detection not 100% reliable!
  - Protocol may miss some errors, but rarely
  - Larger EDC field yields better detection and correction



18

Error Detection - Checksum

- Used by TCP, UDP, IP, etc..
- Ones complement sum of all words/shorts/bytes in packet
- Simple to implement
- Relatively weak detection
  - Easily tricked by typical loss patterns

Example: Internet Checksum

- Goal: detect "errors" (e.g., flipped bits) in transmitted segment

Sender

- Treat segment contents as sequence of 16-bit integers
- Checksum: addition (1's complement sum) of segment contents
- Sender puts checksum value into checksum field in header

Receiver

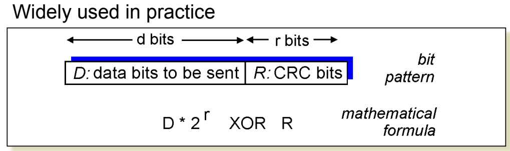
- Compute checksum of received segment
- Check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. But maybe errors nonetheless?

Error Detection – Cyclic Redundancy Check (CRC)

- Polynomial code
  - Treat packet bits a coefficients of n-bit polynomial
  - Choose r+1 bit generator polynomial (well known – chosen in advance)
  - Add r bits to packet such that message is divisible by generator polynomial
- Better loss detection properties than checksums
  - Cyclic codes have favorable properties in that they are well suited for detecting burst errors
  - Therefore, used on networks/hard drives

Error Detection – CRC

- View data bits, D, as a binary number
- Choose r+1 bit pattern (generator), G
- Goal: choose r CRC bits, R, such that
  - <D,R> exactly divisible by G (modulo 2)
  - Receiver knows G, divides <D,R> by G. If non-zero remainder: error detected!
  - Can detect all burst errors less than r+1 bits
- Widely used in practice



CRC Example

Want:

$D \cdot 2^r$ XOR R = nG

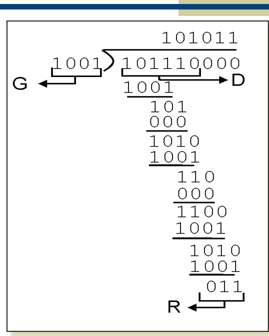
equivalently:

$D \cdot 2^r$ = nG XOR R

equivalently:

if we divide $D \cdot 2^r$ by G, want reminder Rb

$$R = \text{remainder}\left[\frac{D \cdot 2^r}{G}\right]$$

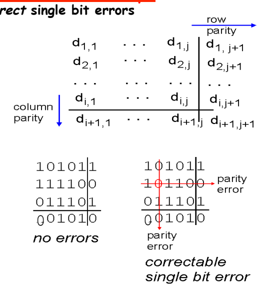


Error Recovery

- Two forms of error recovery
  - Redundancy
    - Error Correcting Codes (ECC)
    - Replication/Voting
  - Retry
- ECC
  - Keep encoded redundant data to help repair losses
  - Forward Error Correction (FEC) – send bits in advance
    - Reduces latency of recovery at the cost of bandwidth

Error Recovery – Error Correcting Codes (ECC)

Two Dimensional Bit Parity:
Detect *and correct* single bit errors

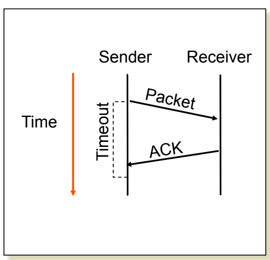


Replication/Voting

- If you take this to the extreme
  [r1] [r2] [r3]
- Send requests to all three versions of the software: Triple modular redundancy
  - Compare the answers, take the majority
  - Assumes no error detection
- In practice - used mostly in space applications; some extreme high availability apps (stocks & banking)? maybe. But usually there are cheaper alternatives if you don't need real-time)
  - Stuff we cover later: surviving malicious failures through voting (byzantine fault tolerance)

Retry – Network Example

- Sometimes errors are transient
- Need to have error detection mechanism
  - E.g., timeout, parity, chksum
  - No need for majority vote



One key question

- How correlated are failures?
- Can you assume independence?
  - If the failure probability of a computer in a rack is p,
  - What is p(computer 2 failing) | computer 1 failed?
    - Maybe it's p... or maybe they're both plugged into the same UPS...
- Why is this important?

Back to Disks… What are our options?

1. Silently return the wrong answer.
2. Detect failure.
   - Every sector has a header with a checksum. Every read fetches both, computes the checksum on the data, and compares it to the version in the header. Returns error if mismatch.
3. Correct / mask the failure
   - Re-read if the firmware signals error (may help if transient error, may not)
   - Use an error correcting code (what kinds of errors do they help?)
     - Bit flips? Yes. Block damaged? No
   - Have the data stored in multiple places (RAID)

Fail-fast disk

```
failfast_get (data, sn) {
    get (s, sn);
    if (checksum(s.data) = s.cksum) {
        data ← s.data;
        return OK;
    } else {
        return BAD;
    }
}
```

Careful disk

```
careful_get (data, sn) {
    r ← 0;
    while (r < 10) {
        r ← failfast_get (data, sn);
        if (r = OK) return OK;
        r++;
    }
    return BAD;
}
```


Fault Tolerant Design

- Quantify probability of failure of each component
- Quantify the costs of the failure
- Quantify the costs of implementing fault tolerance

- This is all probabilities...

32

Summary

- Definition of MTTF/MTBF/MTTR: Understanding availability in systems.
- Failure detection and fault masking techniques
- Engineering tradeoff: Cost of failures vs. cost of failure masking.
  - At what level of system to mask failures?
  - Leading into replication as a general strategy for fault tolerance
- Thought to leave you with:
  - What if you have to survive the failure of entire computers? Of a rack? Of a datacenter?

33

Whole disk replication

- None of these schemes deal with block erasure or disk failure
  - Block erasure: You could do parity on a larger scale. Or you could replicate to another disk. Engineering tradeoff - depends on likelihood of block erasure vs. disk failure; if you have to guard against disk failure already, maybe you don't want to worry as much about large strings of blocks being erased.
- (Gets back to that failure correlation question)

34

Building blocks

- Understand the enemy:
  - Single bit flips (common in memory, sometimes disks, communication channels)
  - Multiple bit flips
  - Block erasure or entire block scrambled
  - Malicious changes vs. accidental
- Checksums - usually used to guard against accidental modification. Example: Parity.
- [0, 1, 0, 1, 0, 1, 1, 1 --> 1] [0, 0, ... -> 0]
  - Weak but fast & easy!
- Or block parity:
  - parity = [block 1] xor [block 2] xor [block 3] ...
- In general: [overhead of checksum] vs [size of blocks] vs [detection power]
- Cryptographic hash functions → usually more expensive, guard against malicious modification
  - Can you see a cool trick you can do with block parity, if you know one component has failed, that you can't do with a hash function? → Error recovery.

Example questions

- You're storing archival data at a bank. The law says you have to keep it for X years. You do not want to mess this up.
- What kinds of failures do you need to deal with?
- What are your options, and what is the cost of those options?
  - error detection, ECC on sector, RAID 1, tape backup, offsite tape backup, etc.
  - Hint: What kind of system-level MTTR can you handle?
- How would your answer change if it was realtime stock trades?

36

"RAID"

- Redundant Array of {Inexpensive, Independent} disks
- Replication! Idea: Write everything to two disks ("RAID-1")
  - If one fails, read from the other
- write(sector, data) ->
  - write(disk1, sector, data)
  - write(disk2, sector, data)
- read(sector, data)
  - data = read(disk1, sector)
  - if error
    - data = read(disk2, sector)
    - if error, return error
  - return data
- Not perfect, though... doesn't solve all uncaught errors.

37

37

more raid

- Option 1: Store a strong checksum with the data to eliminate all uncaught errors
  - Note: In disks today, errors get through checksums. Why?
    - Bits can get flipped at the I/O controller, etc., *after* checksum verification
    - Many checksums aren't 100% strong. If you read 4 trillion sectors with a 1-in-a-million error rate, a 32-bit checksum will let an error through.
      - That would be reading a petabyte of data. That's only 1000 servers reading their entire disk once.

38

38

Durable disk (RAID 1)

```
durable_get (data, sn) {
        r ← disk1.careful_get (data, sn);
        if (r = OK) return OK;
        r ← disk2.careful_get (data, sn);
        signal(repair disk1);
        return r;
}
```

39

If time permits...

- RAID-5

40

40