# Errors and Failures part 1: Local.

Carnegie Mellon 15-440, Distributed Systems

# Last times...

- Started out looking at local methods for mutual exclusion: built up from atomic instructions at the CPU level
- Last two lectures: distributed mutual exclusion. Challenge: No atomic instructions, so we had to talk to everybody.
  - This isn't really unusual: atomic instructions and consistency on a single multi-processor/multi-core machine have to have cache coherence protocols under the covers.
- Importance of timestamps in distributed mutex (and, as we'll see later, many protocols that build on dist. mutex)

# Today

- Quick quiz:
  - What are the most common sources of failure in a computer system?
    - Software bugs...
    - Power supplies
    - Hard drives
    - Memory

And what types of errors?

Hard errors: The component is dead.

Soft errors: A signal or bit is wrong, but it doesn't mean the component is faulty
(Note: You can have recurring soft errors due to faulty, but not dead, h/w)

# 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. :)

# Coping with failures...

A failure

Propagates

- Let's say one bit in your DRAM fails.
- 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.

# Techniques

- We've already seen some: 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 turning you into a hamburger.
- Today: Replication
  - Next time: Specific techniques used in file systems, disks
  - This time: Understand how to quantify reliability
  - Understand basic techniques of replication and fault masking

# Measuring Availability

- Mean time to failure (MTTF)
- Mean time to repair (MTTR)
- Availability = MTTF / (MTTF + MTTR)
- MTBF = MTTF + MTTR
- Suppose OS crashes once per month, takes 10min to reboot. MTTF = 720 hours = 43,200 minutes
  MTTR = 10 minutes
  43200 / 43210 = 0.997 ("3 nines") -- 2 hours downtime per year.

# 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...

# 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%

# 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...



PRODUCT OVERVIEW

## Cheetah 15K.4

Mainstream enterprise disc drive

Simply the best price/ performance, lowest cost of ownership disc drive ever

**KEY FEATURES AND BENEFITS**

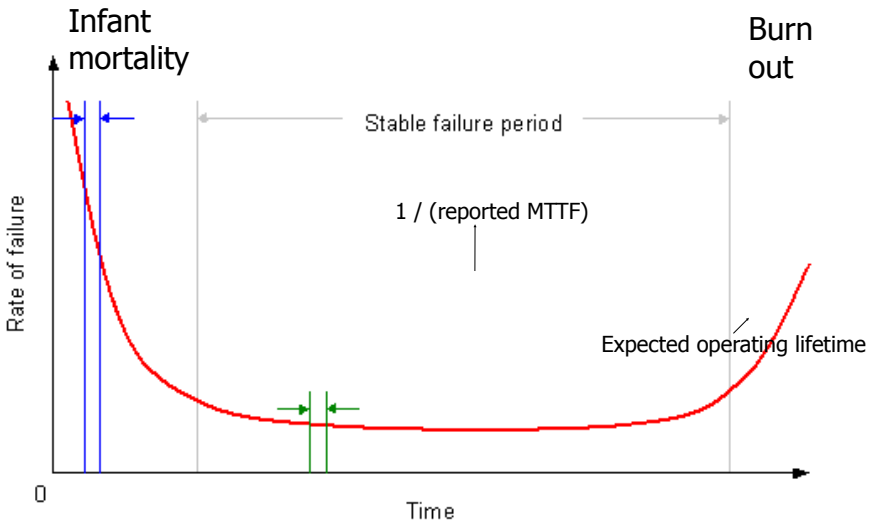
- The Cheetah® 15K.4 is the highest-performance drive ever offered by Seagate®, delivering maximum IOPS with fewer drives to yield lower TCO.
- The Cheetah 15K.4 price-per-performance value united with the breakthrough benefits of serial attached SCSI (SAS) make it the optimal 3.5-inch drive for rock 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**

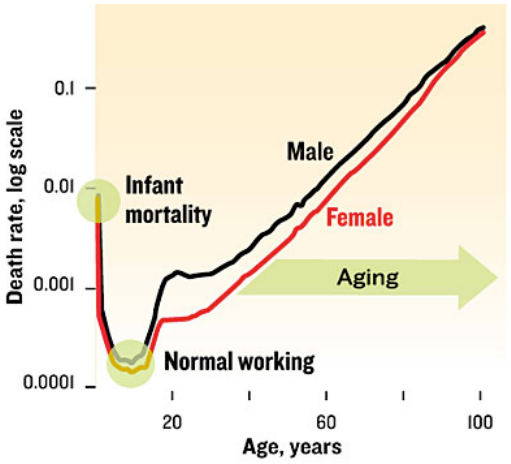
- 146-, 73- and 36-Gbyte capacities
- 3.3-msec average read and 3.8-msec average write seek times
- Up to 96-Mbytes/sec sustained transfer rate
- 1.4 million hours full duty cycle MTBF
- Serial Attached SCSI (SAS), Ultra320 SCSI and 2 Gbits/sec Fibre Channel interfaces
- 5-year warranty

*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




**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.

# 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. [draw pic] 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 (next time)

# 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*.

## 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;
}
```

## Whole disk replication

- None of those schemes dealt 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)

# "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.



# 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.



# 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;
}
```

# If time permits...

- RAID-5

# Voting

- Option 2: 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
- Observe that you can do this at many layers of abstraction!
- 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)
- Really crazy stuff we may hit later: surviving malicious failures through voting (byzantine fault tolerance) -- mostly still theory today; tomorrow, who knows?



# 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?



# Summary of today

- 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?