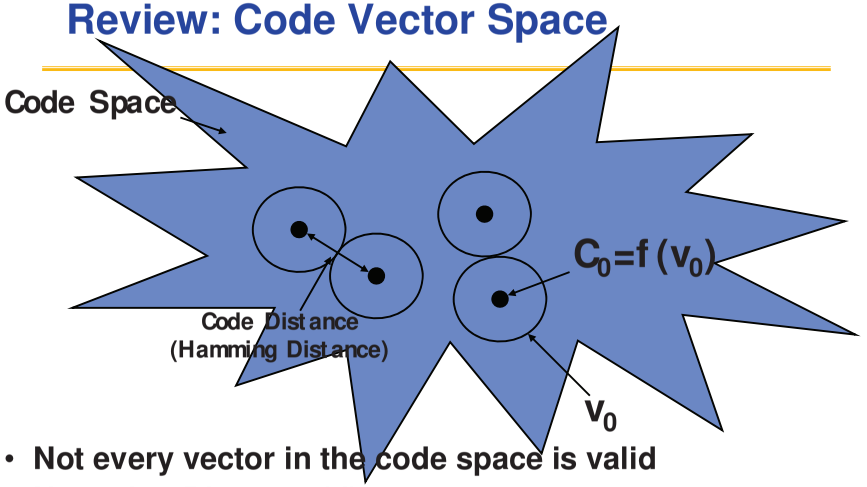
CS252
Graduate Computer Architecture
Lecture 25

Error Correction Codes (Con't)
Disk I/O and Queueing Theory
April 25th, 2012

John Kubiatowicz
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~kubitron/cs252

Review: Code Vector Space



- Not every vector in the code space is valid
- Hamming Distance (d):
  - Minimum number of bit flips to turn one code word into another
- Number of errors that we can detect: (d-1)
- Number of errors that we can fix: ½(d-1)

Recall: Defining Code through H matrix

- Consider a parity-check matrix H (n [n-k])
  - Define valid code words Ci as those that give S_i=0 (null space of H)

$$\overline{S}_i \; ! \; \mathbf{H} \forall \overline{C}_i \; ! \; \overline{0}$$

  - Size of null space?
    (null-rank H)=k if (n-k) linearly independent columns in H
- Suppose we transmit code word C with error:
  - Model this as vector E which flips selected bits of C to get R (received):

$$\overline{R} \; ! \; \overline{C} \# \; \overline{E}$$

  - Consider what happens when we multiply by H:

$$S \; ! \; \mathbf{H} \forall R \; ! \; \mathbf{H} \forall (C \# \; E) \; ! \; \mathbf{H} \forall E$$

- What is distance of code?
  - Code has distance d if no sum of d-1 or less columns yields 0
  - I.e. No error vectors, E, of weight < d have zero syndromes
  - So – Code design is designing H matrix

Recall: How to relate G and H (Binary Codes)

- Defining H makes it easy to understand distance of code, but hard to generate code (H defines code implicitly!)
- However, let H be of following form:

$$\mathbf{H} \; ! \; \exists \mathbf{P} \mid \mathbf{I} \%$$

P is (n-k) k, I is (n-k) (n-k)
Result: H is (n-k) n

- Then, G can be of following form (maximal code size):

$$\mathbf{G} \; ! \; \begin{matrix} +\mathbf{I} \; (\\ ) \\ *\mathbf{P} \; \& \\ \ni \end{matrix}$$

P is (n-k) k, I is k k
Result: G is n k

- Notice: G generates values in null-space of H *and* has k independent columns so generates 2^k unique values:

$$\overline{S}_i \; ! \; \mathbf{H} \forall \exists \mathbf{G} \; \forall w_i \; \% \; 0 \; \begin{matrix}) \\ + \\ ) \\ * \end{matrix} \exists \mathbf{P} \mid \mathbf{I} \% \; \begin{matrix} +\mathbf{I} \; (\; (\; - \\ ) \; \& \& \forall w_i \; , \; 0 \\ *\mathbf{P} \; \& \& \\ \ni \ni \end{matrix}$$

Tweeks:

- No reason cannot make code shorter than maximum
- Suppose n-k=8 bits of parity. What is max code size (n) for d=4?
  - Maximum number of unique, odd-weight columns: $2^7 = 128$
  - So, n = 128. But, then k = n – (n – k) = 120. Weird!
  - Just throw out columns of high weight and make (72, 64) code!
- Circuit optimization: if throwing out column vectors, pick ones of highest weight (# bits=1) to simplify circuit
- Further– shortened codes like this might have d > 4 in some special directions
  - Example: Kaneda paper, catches failures of groups of 4 bits
  - Good for catching chip failures when DRAM has groups of 4 bits
- What about EVENODD code?
  - Can be used to handle two erasures
  - What about two dead DRAMs? Yes, if you can really know they are dead

How to correct errors?

- Consider a parity-check matrix H (n [n-k])
  - Compute the following *syndrome* S_i given code element C_i:

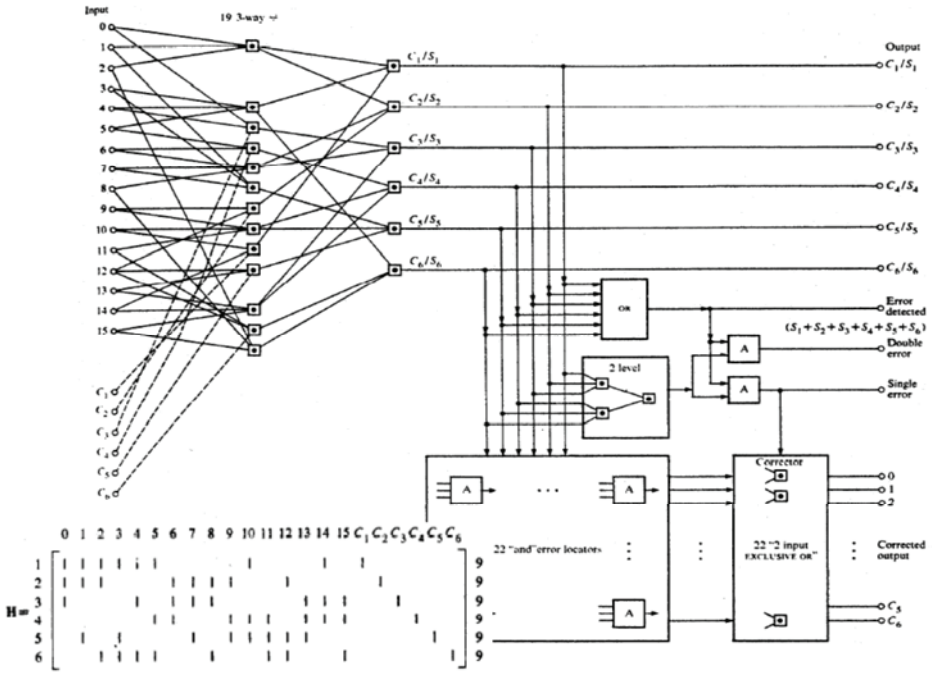
$$\overline{S}_i \# \mathbf{H} \overline{\forall} \overline{C}_i \# \mathbf{H} \overline{\forall} \overline{E}$$

- Suppose that two *correctable* error vectors E_1 and E_2 produce same syndrome:

$$\mathbf{H} \overline{\forall} \overline{E}_1 \# \mathbf{H} \overline{\forall} \overline{E}_2 \ni \mathbf{H} \overline{\forall} \overline{E}_1 \& \overline{E}_2 \not\# 0$$

$$\ni \overline{E}_1 \& \overline{E}_2 \text{ has d or more bits set}$$

- But, since both E_1 and E_2 have ! (d-1)/2 bits set, $E_1 + E_2$! d-1 bits set so this conclusion cannot be true!
- So, syndrome is unique indicator of correctable error vectors



Galois Field

- Definition: *Field:* a complete group of elements with:
  - Addition, subtraction, multiplication, division
  - Completely *closed* under these operations
  - Every element has an additive inverse
  - Every element except zero has a multiplicative inverse
- Examples:
  - Real numbers
  - Binary, called GF(2) (Galois Field with base 2
    - » Values 0, 1. Addition/subtraction: use xor. Multiplicative inverse of 1 is 1
  - Prime field, GF(p) (Galois Field with base p
    - » Values 0 … p-1
    - » Addition/subtraction/multiplication: modulo p
    - » Multiplicative Inverse: every value except 0 has inverse
    - » Example: GF(5): 1 1) 1 mod 5, 2 3) 1mod 5, 4 4) 1 mod 5
  - General Galois Field: GF(p^m) (base p (prime!), dimension m
    - » Values are vectors of elements of GF(p) of dimension m
    - » Add/subtract: vector addition/subtraction
    - » Multiply/divide: more complex
    - » Just like read numbers but finite!
    - » Common for computer algorithms: GF(2^m)

Specific Example: Galois Fields GF(2^n)

- Consider *polynomials* whose coefficients come from GF(2).
- Each term of the form x^n is either present or absent.
- Examples: $0, 1, x, x^2$, and
  $x^7 + x^6 + 1 = 1 \cdot x^7 + 1 \cdot x^6 + 0 \cdot x^5 + 0 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0$
- With addition and multiplication these form a "ring" (not quite a field – still missing division):
- "Add": XOR each element individually with no carry:

$$\begin{array}{r} x^4 + x^3 + \quad + x + 1 \\ +\quad x^4 + \quad + x^2 + x \\ \hline x^3 + x^2 \quad + 1 \end{array}$$

- "Multiply": multiplying by x is like shifting to the left.

$$\begin{array}{r} x^2 + x + 1 \\ x + 1 \\ \hline x^2 + x + 1 \\ x^3 + x^2 + x \\ x^3 \quad + 1 \end{array}$$

So what about division (mod)

$\frac{x^4 + x^2}{x} = x^3 + x$ with remainder 0

$\frac{x^4 + x^2 + 1}{x + 1} = x^3 + x^2$ with remainder 1

$$\begin{array}{r|l} & x^3 + x^2 + 0x + 0 \\ \hline x + 1 & x^4 + 0x^3 + x^2 + 0x + 1 \\ & x^4 + x^3 \\ \hline & x^3 + x^2 \\ & x^3 + x^2 \\ \hline & 0x^2 + 0x \\ & 0x + 1 \end{array}$$

Remainder 1

Producing Galois Fields

- These polynomials form a Galois (finite) field if we take the results of this multiplication modulo a prime polynomial p(x)
  - A prime polynomial cannot be written as product of two non-trivial polynomials q(x)r(x)
  - For any degree, there exists at least one prime polynomial.
  - With it we can form GF(2n)
- Every Galois field has a primitive element, α, such that all non-zero elements of the field can be expressed as a power of α
  - Certain choices of $p(x)$ make the simple polynomial x the primitive element. These polynomials are called *primitive*
- For example, $x^4 + x + 1$ is primitive. So $\alpha = x$ is a primitive element and successive powers of α will generate all non-zero elements of GF(16).
- *Example on next slide.*

Galois Fields with primitive $x^4 + x + 1$

$$\begin{aligned} \alpha^0 &= 1 \\ \alpha^1 &= x \\ \alpha^2 &= x^2 \\ \alpha^3 &= x^3 \\ \alpha^4 &= x + 1 \\ \alpha^5 &= x^2 + x \\ \alpha^6 &= x^3 + x^2 \\ \alpha^7 &= x^3 + x + 1 \\ \alpha^8 &= x^2 + 1 \\ \alpha^9 &= x^3 + x \\ \alpha^{10} &= x^2 + x + 1 \\ \alpha^{11} &= x^3 + x^2 + x \\ \alpha^{12} &= x^3 + x^2 + x + 1 \\ \alpha^{13} &= x^3 + x^2 + 1 \\ \alpha^{14} &= x^3 + 1 \\ \alpha^{15} &= 1 \end{aligned}$$

- Primitive element = x in GF(2^n)

$$\begin{aligned} \alpha^4 &= x^4 \bmod x^4 + x + 1 \\ &= x^4 \text{ xor } x^4 + x + 1 \\ &= x + 1 \end{aligned}$$

- In general finding primitive polynomials is difficult. Most people just look them up in a table, such as:

Primitive Polynomials

$x^2 + x + 1$
$x^3 + x + 1$
$x^4 + x + 1$
$x^5 + x^2 + 1$
$x^6 + x + 1$
$x^7 + x^3 + 1$
$x^8 + x^4 + x^3 + x^2 + 1$
$x^9 + x^4 + 1$
$x^{10} + x^3 + 1$
$x^{11} + x^2 + 1$
$x^{12} + x^6 + x^4 + x + 1$
$x^{13} + x^4 + x^3 + x + 1$
$x^{14} + x^{10} + x^6 + x + 1$
$x^{15} + x + 1$
$x^{16} + x^{12} + x^3 + x + 1$
$x^{17} + x^3 + 1$
$x^{18} + x^7 + 1$
$x^{19} + x^5 + x^2 + x + 1$
$x^{20} + x^3 + 1$
$x^{21} + x^2 + 1$
$x^{22} + x + 1$
$x^{23} + x^5 + 1$
$x^{24} + x^7 + x^2 + x + 1$
$x^{25} + x^3 + 1$
$x^{26} + x^6 + x^2 + x + 1$
$x^{27} + x^5 + x^2 + x + 1$
$x^{28} + x^3 + 1$
$x^{29} + x + 1$
$x^{30} + x^6 + x^4 + x + 1$
$x^{31} + x^3 + 1$
$x^{32} + x^7 + x^6 + x^2 + 1$

Galois Field	Hardware
Multiplication by x	shift left
Taking the result mod $p(x)$	XOR-ing with the coefficients of $p(x)$ when the most significant coefficient is 1.
Obtaining all 2^n-1 non-zero elements by evaluating x^k for $k = 1, \ldots, 2^n-1$	Shifting and XOR-ing 2^n-1 times.

Reed-Solomon Codes

- Galois field codes: code words consist of *symbols*
  - Rather than bits
- Reed-Solomon codes:
  - Based on polynomials in GF(2^k) (I.e. k-bit symbols)
  - Data as coefficients, code space as values of polynomial:
  - $P(x)=a_0+a_1x^1+\ldots a_{k-1}x^{k-1}$
  - Coded: P(0),P(1),P(2)….,P(n-1)
  - Can recover polynomial as long as get *any* k of n
- Properties: can choose number of check symbols
  - Reed-Solomon codes are "maximum distance separable" (MDS)
  - Can add d symbols for distance d+1 code
  - Often used in "erasure code" mode: as long as no more than n-k coded symbols erased, can recover data
- Side note: Multiplication by constant in GF(2^k) can be represented by k! k matrix: a∀x
  - Decompose unknown vector into k bits: $x=x_0+2x_1+\ldots+2^{k-1}x_{k-1}$
  - Each column is result of multiplying a by 2^i

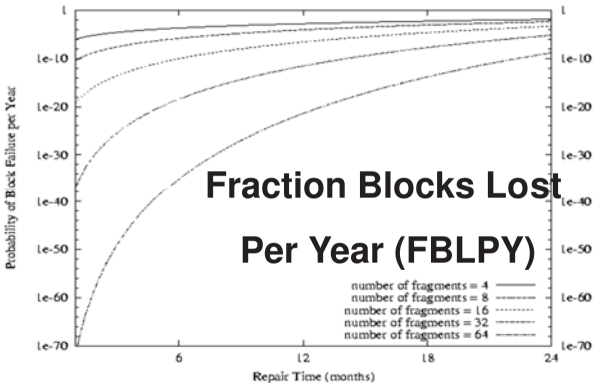
Reed-Solomon Codes (con't)

- Reed-solomon codes (Non-systematic):
  - Data as coefficients, code space as values of polynomial:
  - $P(x)=a_0+a_1x^1+\ldots a_6x^6$
  - Coded: P(0),P(1),P(2)….,P(6)
- Called Vandermonde Matrix: maximum rank
- Different representation (This H' and G not related)
  - Clear that all combinations of two or less columns independent d=3
  - Very easy to pick whatever d you happen to want: add more rows
- Fast, Systematic version of Reed-Solomon:
  - Cauchy Reed-Solomon, others

$$G = \begin{pmatrix} 1^0 & 1^1 & 1^2 & 1^3 & 1^4 \\ 2^0 & 2^1 & 2^2 & 2^3 & 2^4 \\ 3^0 & 3^1 & 3^2 & 3^3 & 3^4 \\ 4^0 & 4^1 & 4^2 & 4^3 & 4^4 \\ 5^0 & 5^1 & 5^2 & 5^3 & 5^4 \\ 6^0 & 6^1 & 6^2 & 6^3 & 6^4 \\ 7^0 & 7^1 & 7^2 & 7^3 & 7^4 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}$$

$$H' = \begin{pmatrix} 1^0 & 2^0 & 3^0 & 4^0 & 5^0 & 6^0 & 7^0 \\ 1^1 & 2^1 & 3^1 & 4^1 & 5^1 & 6^1 & 7^1 \end{pmatrix}$$

Aside: Why erasure coding? High Durability/overhead ratio!

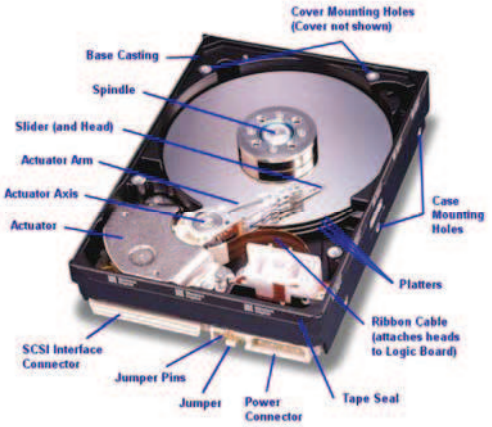


- Exploit law of large numbers for durability!
- 6 month repair, FBLPY:
  - Replication: 0.03
  - Fragmentation: 10^{-35}

Motivation: Who Cares About I/O?

- CPU Performance: 60% per year
- I/O system performance limited by mechanical delays (disk I/O) or time to access remote services
  - Improvement of < 10% per year (IO per sec or MB per sec)
- Amdahl's Law: system speed-up limited by the slowest part!
  - 10% IO & 10x CPU => 5x Performance (lose 50%)
  - 10% IO & 100x CPU => 10x Performance (lose 90%)
- I/O bottleneck:
  - Diminishing fraction of time in CPU
  - Diminishing value of faster CPUs

Hard Disk Drives



Western Digital Drive

http://www.storagereview.com/guide/

Read/Write Head

Side View

IBM/Hitachi Microdrive

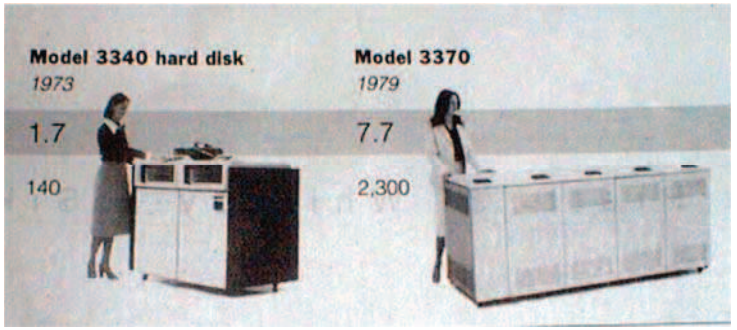
Historical Perspective

- 1956 IBM Ramac — early 1970s Winchester
  - Developed for mainframe computers, proprietary interfaces
  - Steady shrink in form factor: 27 in. to 14 in.
- Form factor and capacity drives market more than performance
- 1970s developments
  - 5.25 inch floppy disk formfactor (microcode into mainframe)
  - Emergence of industry standard disk interfaces
- Early 1980s: PCs and first generation workstations
- Mid 1980s: Client/server computing
  - Centralized storage on file server
    - » accelerates disk downsizing: 8 inch to 5.25
  - Mass market disk drives become a reality
    - » industry standards: SCSI, IPI, IDE
    - » 5.25 inch to 3.5 inch drives for PCs, End of proprietary interfaces
- 1900s: Laptops => 2.5 inch drives
- 2000s: Shift to perpendicular recording
  - 2007: Seagate introduces 1TB drive
  - 2009: Seagate/WD introduces 2TB drive

Disk History

Data density Mbit/sq. in.

Capacity of Unit Shown Megabytes

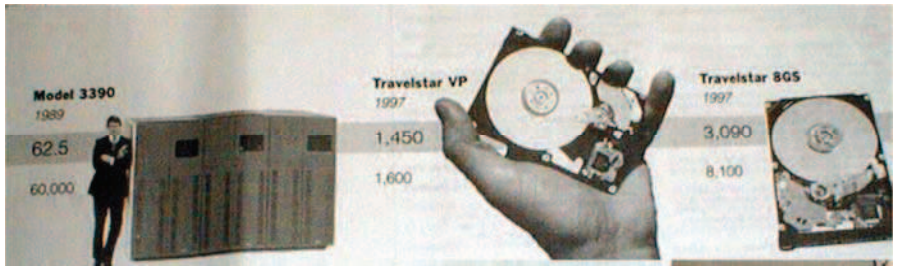


1973: 1. 7 Mbit/sq. in 140 MBytes

1979: 7. 7 Mbit/sq. in 2,300 MBytes

source: New York Times, 2/23/98, page C3, "Makers of disk drives crowd even mroe data into even smaller spaces"

Disk History



1989:
63 Mbit/sq. in
60,000 MBytes

1997:
1450 Mbit/sq. in
2300 MBytes

1997:
3090 Mbit/sq. in
8100 MBytes

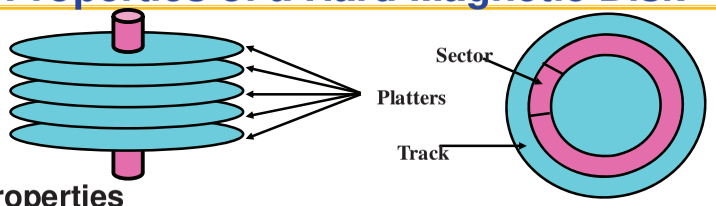
source: New York Times, 2/23/98, page C3,
"Makers of disk drives crowd even mroe data into even smaller spaces"

Example: Seagate Barracuda (2010)

- 3TB! 488 Gb/in^2
- 5 (3.5") platters, 2 heads each
- Perpendicular recording
- 7200 RPM, 4.16ms latency
- 600MB/sec burst,
  149MB/sec sustained transfer speed
- 64MB cache
- Error Characteristics:
  - MBTF: 750,000 hours
  - Bit error rate: 10^{-14}
- Special considerations:
  - Normally need special "bios" (EFI): Bigger than easily handled by 32-bit OSes.
  - Seagate provides special "Disk Wizard" software that virtualizes drive into multiple chunks that makes it bootable on these OSes.

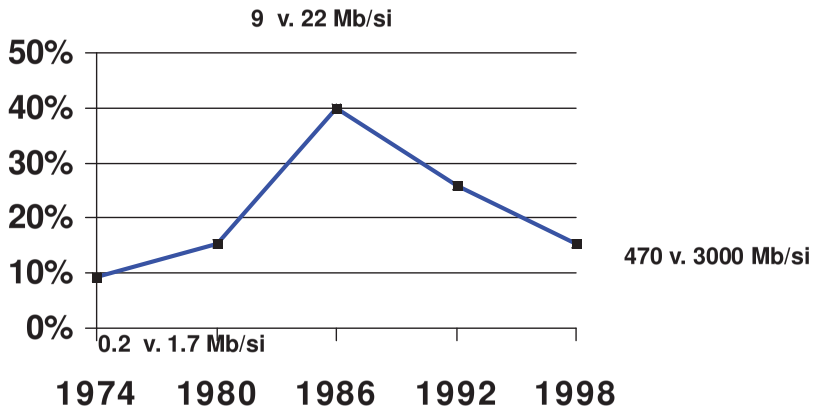
4/25/2012 cs252-S12, Lecture 25 22

Properties of a Hard Magnetic Disk



- Properties
  - Independently addressable element: sector
    - » OS always transfers groups of sectors together—"blocks"
  - A disk can access directly any given block of information it contains (random access). Can access any file either sequentially or randomly.
  - A disk can be rewritten in place: it is possible to read/modify/write a block from the disk
- Typical numbers (depending on the disk size):
  - 500 to more than 20,000 tracks per surface
  - 32 to 800 sectors per track
    - » A sector is the smallest unit that can be read or written
- Zoned bit recording
  - Constant bit density: more sectors on outer tracks
  - Speed varies with track location

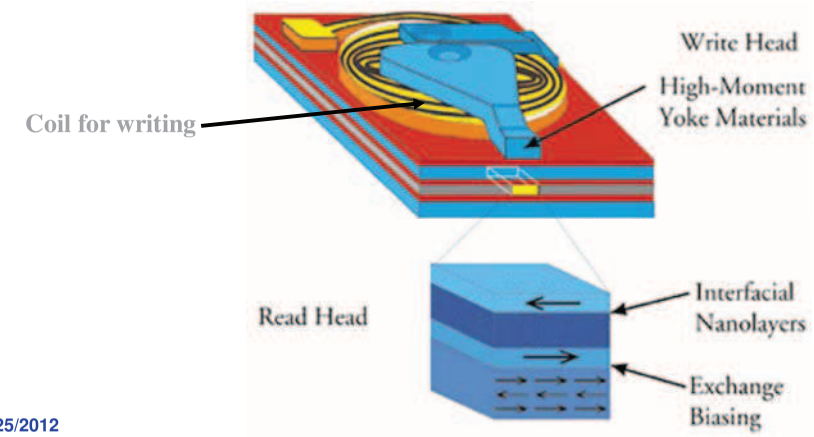
MBits per square inch: DRAM as % of Disk over time



source: New York Times, 2/23/98, page C3,
"Makers of disk drives crowd even mroe data into even smaller spaces"

Nano-layered Disk Heads

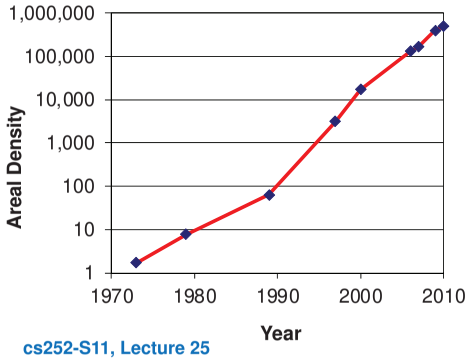
- Special sensitivity of Disk head comes from "Giant Magneto-Resistive effect" or (GMR)
- IBM is (was) leader in this technology
  - Same technology as TMJ-RAM breakthrough



Disk Figure of Merit: Areal Density

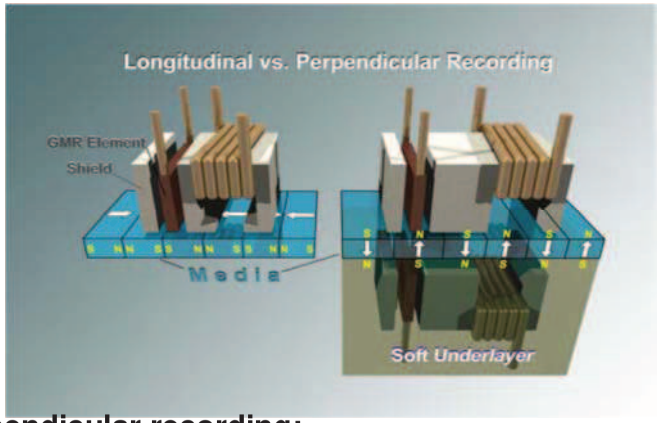
- Bits recorded along a track
  - Metric is Bits Per Inch (BPI)
- Number of tracks per surface
  - Metric is Tracks Per Inch (TPI)
- Disk Designs Brag about bit density per unit area
  - Metric is Bits Per Square Inch: Areal Density = BPI x TPI

Year	Areal Density
1973	2
1979	8
1989	63
1997	3,090
2000	17,100
2006	130,000
2007	164,000
2009	400,000
2010	488,000



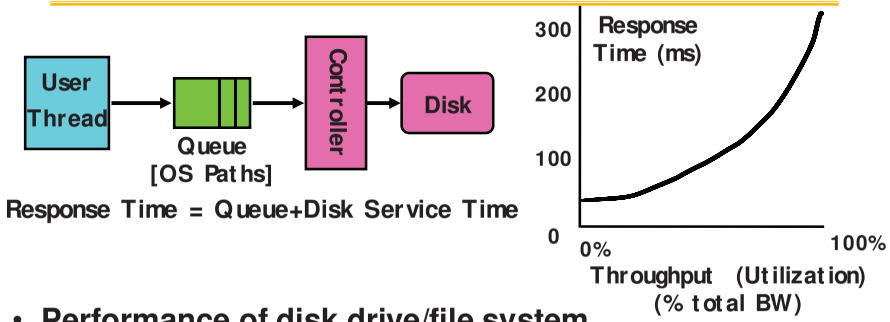
4/25/2012 26

Newest technology: Perpendicular Recording



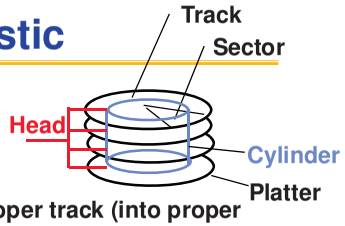
- In Perpendicular recording:
  - Bit densities much higher
  - Magnetic material placed on top of magnetic underlayer that reflects recording head and effectively doubles recording field

Disk I/O Performance

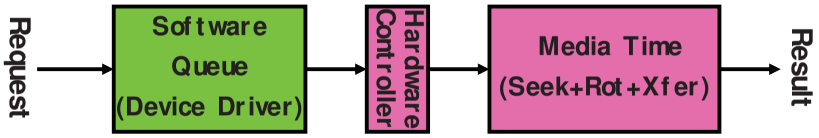


- Performance of disk drive/file system
  - Metrics: Response Time, Throughput
  - Contributing factors to latency:
    - » Software paths (can be loosely modeled by a queue)
    - » Hardware controller
    - » Physical disk media
- Queuing behavior:
  - Can lead to big increase of latency as utilization approaches 100%

Magnetic Disk Characteristic



- Cylinder: all the tracks under the head at a given point on all surface
- Read/write data is a three-stage process:
  - Seek time: position the head/arm over the proper track (into proper cylinder)
  - Rotational latency: wait for the desired sector to rotate under the read/write head
  - Transfer time: transfer a block of bits (sector) under the read-write head
- Disk Latency = Queueing Time + Controller time + Seek Time + Rotation Time + Xfer Time



- Highest Bandwidth:
  - transfer large group of blocks sequentially from one track

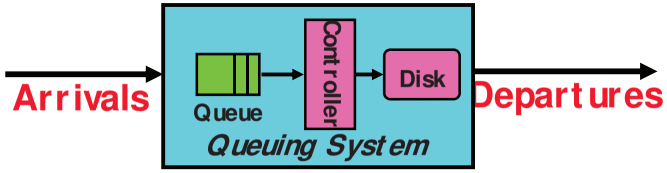
Disk Time Example

- Disk Parameters:
  - Transfer size is 8K bytes
  - Advertised average seek is 12 ms
  - Disk spins at 7200 RPM
  - Transfer rate is 4 MB/sec
- Controller overhead is 2 ms
- Assume that disk is idle so no queuing delay
- Disk Latency =
  Queuing Time +
  Seek Time + Rotation Time + Xfer Time + Ctrl Time
- What is Average Disk Access Time for a Sector?
  - Ave seek + ave rot delay + transfer time + controller overhead
  - 12 ms + [0.5/(7200 RPM/60s/M)] 1000 ms/s + [8192 bytes/(4 10^6 bytes/s)] 1000 ms/s + 2 ms
  - 12 + 4.17 + 2.05 + 2 = 20.22 ms
- Advertised seek time assumes no locality: typically 1/4 to 1/3 advertised seek time: 12 ms => 4 ms

Typical Numbers of a Magnetic Disk

- Average seek time as reported by the industry:
  - Typically in the range of 4 ms to 12 ms
  - Due to locality of disk reference may only be 25% to 33% of the advertised number
- Rotational Latency:
  - *Most* disks rotate at 3,600 to 7200 RPM (Up to 15,000RPM or more)
  - Approximately 16 ms to 8 ms per revolution, respectively
  - An average latency to the desired information is halfway around the disk: 8 ms at 3600 RPM, 4 ms at 7200 RPM
- Transfer Time is a function of:
  - Transfer size (usually a sector): 1 KB / sector
  - Rotation speed: 3600 RPM to 15000 RPM
  - Recording density: bits per inch on a track
  - Diameter: ranges from 1 in to 5.25 in
  - Typical values: 2 to 50 MB per second
- Controller time?
  - Depends on controller hardware—need to examine each case individually

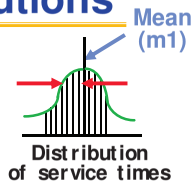
Introduction to Queuing Theory

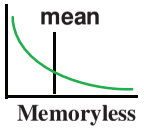


- What about queuing time??
  - Let's apply some queuing theory
  - Queuing Theory applies to long term, steady state behavior ! Arrival rate = Departure rate
- Little's Law:
  Mean # tasks in system = arrival rate x mean response time
  - Observed by many, Little was first to prove
  - Simple interpretation: you should see the same number of tasks in queue when entering as when leaving.
- Applies to any system in equilibrium, as long as nothing in black box is creating or destroying tasks
  - Typical queuing theory doesn't deal with transient behavior, only steady-state behavior

Background: Use of random distributions

- Server spends variable time with customers
  - Mean (Average) m1 = p(T)! T
  - Variance ∀² = p(T)! (T-m1)² = p(T)! T²-m1=E(T²)-m1
  - Squared coefficient of variance: C = ∀²/m1² Aggregate description of the distribution.
- Important values of C:
  - No variance or deterministic # C=0
  - "memoryless" or exponential # C=1
    - » Past tells nothing about future
    - » Many complex systems (or aggregates) well described as memoryless
  - Disk response times C ∃ 1.5 (majority seeks < avg)
- Mean Residual Wait Time, m1(z):
  - Mean time must wait for server to complete current task
  - Can derive m1(z) = ½m1! (1 + C)
    - » Not just ½m1 because doesn't capture variance
  - C = 0 # m1(z) = ½m1; C = 1 # m1(z) = m1



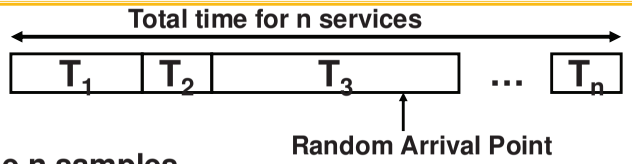


A Little Queuing Theory: Mean Wait Time

- Parameters that describe our system:
  - %: mean number of arriving customers/second
  - T_{ser}: mean time to service a customer ("m1")
  - C: squared coefficient of variance = ∀²/m1²
  - : service rate = $1/T_{ser}$
  - u: server utilization (0&u&1): u = % = %! T_{ser}
- Parameters we wish to compute:
  - T_q: Time spent in queue
  - L_q: Length of queue = %! T_q (by Little's law)
- Basic Approach:
  - Customers before us must finish; mean time = L_q ! T_{ser}
  - If something at server, takes m1(z) to complete on avg
    - » Chance server busy = u # mean time is u ! m1(z)
- Computation of wait time in queue (T_q):

  $T_q = L_q$! T_{ser} + u ! m1(z)

Mean Residual Wait Time: m1(z)



- Imagine n samples
  - There are *n ! P(Tx)* samples of size T_x
  - Total space of samples of size T_x: $T_x ! n ! P(T_x) (n ! T_x P(T_x)$
  - Total time for n services: $) _x n ! T_x P(T_x) (n ! T_{ser}$
  - Chance arrive in service of length T_x: $\frac{n ! T_x P(T_x)}{n ! T_{ser}} (\frac{T_x P(T_x)}{T_{ser}}$
  - Avg remaining time if land in T_x: $½T_x$
  - Finally: Average Residual Time m1(z):

$$) _x \frac{21}{012} T_x / \frac{2 T_x P(T_x) /}{0 \cdot 1 T_{ser}} = (\frac{1}{2} \frac{E(T^2)}{T_{ser}} (\frac{1}{2} T_{ser} \frac{2 \forall^2 , T_{ser}{}^2 /}{0 1 T_{ser}{}^2} = (\frac{1}{2} T_{ser} * 1 , C +$$

A Little Queuing Theory: M/G/1 and M/M/1

- Computation of wait time in queue (T_q):

  $T_q = L_q$! T_{ser} + u ! m1(z)

  Little's Law

  T_q = %! T_q ! T_{ser} + u ! m1(z)

  T_q = u ! T_q + u ! m1(z)

  Defn of utilization (u)

  T_q ! (1 – u) = m1(z) ! u # T_q = m1(z) ! u/(1-u) #

  $T_q = T_{ser}$! ½(1+C) ! u/(1 – u)
- Notice that as u3 1, T_q3 4 !
- Assumptions so far:
  - System in equilibrium; No limit to the queue: works First-In-First-Out
  - Time between two successive arrivals in line are random and memoryless: (M for C=1 exponentially random)
  - Server can start on next customer immediately after prior finishes
- General service distribution (no restrictions), 1 server:
  - Called M/G/1 queue: $T_q = T_{ser}$ x ½(1+C) x u/(1 – u))
- Memoryless service distribution (C = 1):
  - Called M/M/1 queue: $T_q = T_{ser}$ x u/(1 – u)

A Little Queuing Theory: An Example

- Example Usage Statistics:
  - User requests 10 x 8KB disk I/Os per second
  - Requests & service exponentially distributed (C=1.0)
  - Avg. service = 20 ms (From controller+seek+rot+trans)
- Questions:
  - How utilized is the disk?
    - » Ans: server utilization, $u = T_{ser}$
  - What is the average time spent in the queue?
    - » Ans: T_q
  - What is the number of requests in the queue?
    - » Ans: L_q
  - What is the avg response time for disk request?
    - » Ans: $T_{sys} = T_q + T_{ser}$
- Computation:

(avg # arriving customers/s) = 10/s

T_{ser} (avg time to service customer) = 20 ms (0.02s)

u (server utilization) = x T_{ser}= 10/s x .02s = 0.2

T_q (avg time/customer in queue) = T_{ser} x u/(1 – u)
= 20 x 0.2/(1-0.2) = 20 x 0.25 = 5 ms (0 .005s)

L_q (avg length of queue) = x T_q=10/s x .005s = 0.05

T_{sys} (avg time/customer in system) =T_q + T_{ser}= 25 ms

Conclusion

- ECC: add redundancy to correct for errors
  - (n,k,d) ! n code bits, k data bits, distance d
  - Linear codes: code vectors computed by linear transformation
- Erasure code: after identifying "erasures", can correct
- Reed-Solomon codes
  - Based on GF(p^n), often GF(2^n)
  - Easy to get distance d+1 code with d extra symbols
  - Often used in erasure mode
- Disk Time = queue + controller + seek + rotate + transfer
- Advertised average seek time benchmark much greater than average seek time in practice
- Queueing theory: $W = \left(\frac{1}{2}\left(1 + C\right)\bar{x}u\right)\Big/\left(1 * u\right)$ for (c=1): $W = \left(\frac{\bar{x}u}{1 * u}\right)$