Error Correction Codes (Con’t)  
Disk I/O and Queueing Theory  
April 26th, 2010  

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

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

**Recall: Defining Code through H matrix**
- Consider a parity-check matrix $H$ (n×[n-k])
  - Define valid code words $C_i$ as those that give $S_i=0$ (null space of $H$)
    \[ S_i = H \cdot C_i = 0 \]
  - 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):
    \[ R = C \oplus E \]
  - Consider what happens when we multiply by $H$:
    \[ S = H \cdot R = H \cdot (C \oplus E') = H \cdot E \]
- 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

**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:
  \[ H = \begin{pmatrix} P \\ I \end{pmatrix} \]
  - $P$ 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):
  \[ G = \begin{pmatrix} I \\ P \end{pmatrix} \]
  - $P$ 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:
  \[ S_i = H \cdot (G \cdot \bar{v}_i) = \left( \begin{pmatrix} P \\ I \end{pmatrix} \cdot \begin{pmatrix} I \\ P \end{pmatrix} \right) \cdot \bar{v}_i \equiv 0 \]
Simple example (Parity, d=2)
- Parity code (8-bits):
  \[
  G = \begin{pmatrix}
  1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
  0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
  0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
  0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
  0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
  0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
  1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 
  \end{pmatrix}
  \]
- Note: Complexity of logic depends on number of 1s in row!

Simple example: Repetition (voting, d=3)
- Repetition code (1-bit):
  \[
  \begin{align*}
  G &= \begin{pmatrix}
  1 \\
  1 \\
  \end{pmatrix} \\
  H &= \begin{pmatrix}
  1 & 1 & 0 \\
  1 & 0 & 1 
  \end{pmatrix}
  \]
- Positives: simple
- Negative: Expensive: only 33% of code word is data

Example: Hamming Code (d=3)
- Binary Hamming code meets Hamming bound
- Recall bound for d=3:
  \[2^k \cdot (1+n) \leq 2^n \Rightarrow n \leq 2^{n-k} - 1\]
- So, rearranging:
  \[k \leq 2^c - (c+1), c = n - k\]
- Thus, for:
  - c=2 check bits, k \leq 4 (Repetition code)
  - c=3 check bits, k \leq 5
  - c=4 check bits, k \leq 7
  - c=5 check bits, k \leq 11
  - c=6 check bits, k \leq 15
  - c=7 check bits, k \leq 23
- H matrix consists of all unique, non-zero vectors
  \[
  H = \begin{pmatrix}
  1 & 0 & 1 & 1 & 1 & 0 & 0 \\
  1 & 1 & 0 & 1 & 0 & 1 & 0 \\
  1 & 1 & 1 & 0 & 0 & 0 & 1 
  \end{pmatrix}
  \]

Example, d=4 code (SEC-DED)
- Design H with:
  - All columns non-zero, odd-weight, distinct
    » Note that odd-weight refers to Hamming Weight, i.e. number of zeros
- Why does this generate d=4?
  - Any single bit error will generate a distinct, non-zero value
  - Any double error will generate a distinct, non-zero value
    » Why? Add together two distinct columns, get distinct result
  - Any triple error will generate a non-zero value
    » Why? Add together three odd-weight values, get an odd-weight value
  - So: need four errors before indistinguishable from code word
- Because d=4:
  - Can correct 1 error (Single Error Correction, i.e. SEC)
  - Can detect 2 errors (Double Error Detection, i.e. DED)
- Example:
  - Note: log size of nullspace will be (columns – rank) = 4, so:
    » Rank = 4, since rows independent, 4 cols indpt
    » Clearly, 8 bits in code word
    » Thus: (8,4) code
Tweeks:

- No reason cannot make code shorter than required
- 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 $C_i$:
    \[
    \bar{S}_i = H \cdot \bar{C}_i = H \cdot \bar{E}
    \]
  
  - Suppose that two correctable error vectors $E_1$ and $E_2$ produce same syndrome:
    \[
    H \cdot \bar{E}_1 = H \cdot \bar{E}_2 \Rightarrow H \left((\bar{E}_1 + \bar{E}_2)\right) = 0
    \]
    \[
    \Rightarrow \bar{E}_1 + \bar{E}_2 \text{ has d or more bits set}
    \]
  
  - But, since both $E_1$ and $E_2$ have $\leq (d-1)/2$ bits set, $E_1 + E_2 \leq d-1$ 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) \(\subseteq\) Galois Field with base 2
    - Values 0, 1. Addition/subtraction: use xor. Multiplicative inverse of 1 is 1
  - Prime field, GF(p) \(\subseteq\) 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\times 1 \equiv 1 \mod 5, 2\times 3 = 1\mod 5, 4\times 4 = 1 \mod 5
  - General Galois Field: GF(p^n) \(\subseteq\) 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 real 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 $x^3 + x^d + 1$
  
  \[ 1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x^1 + 0 \cdot x^0 = x^3 + x^2 + 0 \cdot x^1 + 0 \cdot x^0 + 1 \cdot x^0 \]
- With addition and multiplication these form a "ring" (not quite a field – still missing division):
  - **Add**: XOR each element individually with no carry:
    \[
    \begin{array}{c}
    x^4 + x^2 + x + 1 \\
    + x^4 + x^2 + x \\
    \hline
    x^3 + x^2 + 1
    \end{array}
    \]
  - **Multiply**: multiplying by $x$ is like shifting to the left.

So what about division (mod)

\[
\frac{x^4 + x^2}{x} = x^3 + x \quad \text{with remainder} \ 0
\]

\[
\frac{x^4 + x^2 + 1}{x + 1} = x^3 + x^2 \quad \text{with remainder} \ 1
\]

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(2^n)
- Every Galois field has a primitive element, $\alpha$, such that all non-zero elements of the field can be expressed as a power of $\alpha$
  - 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 $\alpha$ will generate all non-zero elements of GF(16).
  - Example on next slide.

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

\[
\begin{align*}
\alpha^0 &= 1 \\
\alpha^1 &= x \\
\alpha^2 &= x^2 \\
\alpha^3 &= x^3 \\
\alpha^4 &= x^4 \\
\alpha^5 &= x^5 \\
\alpha^6 &= x^6 \\
\alpha^7 &= x^7 \\
\alpha^8 &= x^8 \\
\alpha^9 &= x^9 \\
\alpha^{10} &= x^{10} \\
\alpha^{11} &= x^{11} \\
\alpha^{12} &= x^{12} \\
\alpha^{13} &= x^{13} \\
\alpha^{14} &= x^{14} \\
\alpha^{15} &= x^{15}
\end{align*}
\]

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

\[
\begin{align*}
\alpha^4 &= x^4 \mod x^4 + x + 1 \\
&= x^4 \text{ xor } x^4 + x + 1 \\
&= x + 1
\end{align*}
\]

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

\[
\begin{align*}
\alpha^4 &= x^4 \mod x^4 + x + 1 \\
&= x^4 \text{ xor } x^4 + x + 1 \\
&= x + 1
\end{align*}
\]
### Primitive Polynomials

- \(x^2 + x + 1\)
- \(x^3 + x + 1\)
- \(x^4 + x + 1\)
- \(x^5 + x^2 + 1\)
- \(x^6 + x^4 + x^3 + x + 1\)
- \(x^8 + x^5 + x^3 + x + 1\)
- \(x^{10} + x^5 + x^3 + x + 1\)
- \(x^{11} + x^7 + x^6 + x^2 + 1\)

### Reed-Solomon Codes

- **Galois field codes**: code words consist of symbols
  - Rather than bits
- **Reed-Solomon codes**:
  - Based on polynomials in \(\mathbb{GF}(2^k)\) (i.e. \(k\)-bit symbols)
  - Data as coefficients, code space as values of polynomial:
    - \(P(x)=a_0+a_1x+a_2x^2+...+a_{k-1}x^{k-1}\)
    - 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 \(\mathbb{GF}(2^k)\) can be represented
  - by \(k\times k\) matrix: \(a\cdot x\)
    - Decompose unknown vector into \(k\) bits: \(x=x_0+2x_1+...+2^{k-1}x_{k-1}\)
    - 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:
    - \(P(x)=a_0+a_1x+a_2x^2+...+a_{k-1}x^{k-1}\)
    - 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 - \(d\leq 3\)
  - Very easy to pick whatever \(d\) you happen to want: add more rows
- **Fast, Systematic version of Reed-Solomon:**
  - Cauchy Reed-Solomon, others

### Aside: Why erasure coding?

**High Durability/overhead ratio!**

**Fraction Blocks Lost**

Per Year (FBLPY)

- **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 (I/O per sec or MB per sec)
- Amdahl’s Law: system speed-up limited by the slowest part!
  - 10% I/O & 10x CPU => 5x Performance (lose 50%)
  - 10% I/O & 100x CPU => 10x Performance (lose 90%)
- I/O bottleneck:
  - Diminishing fraction of time in CPU
  - Diminishing value of faster CPUs

A Three-Bus System (+ backside cache)

- A small number of backplane buses tap into the processor-memory bus
  - Processor-memory bus is only used for processor-memory traffic
  - I/O buses are connected to the backplane bus
- Advantage: loading on the processor/memory bus is greatly reduced ⇒ Faster access to memory

Main components of Intel Chipset: Pentium 4

- Northbridge:
  - Handles memory
  - Graphics
- Southbridge: I/O
  - PCI bus
  - Disk controllers
  - USB controllers
  - Audio
  - Serial I/O
  - Interrupt controller
  - Timers

Hard Disk Drives

- Read/Write Head Side View
- Western Digital Drive
  http://www.storagereview.com/guide/
- 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.
- **1970s developments**
  - 5.25 inch floppy disk form factor (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
- **1990s: Laptops => 2.5 inch drives**
- **2000s: Shift to perpendicular recording**
  - 2007: Seagate introduces 1TB drive
  - 2009: Seagate/WD promises 2TB drive

**Disk History**

- **1973:**
  - 1.7 Mbit/sq. in
  - 140 MBytes
- **1979:**
  - 7.7 Mbit/sq. in
  - 2,300 MBytes


**Seagate Barracuda (2009)**

- 2TB! 400 GB/in²
- 4 platters, 2 heads each
- 3.5” platters
- Perpendicular recording
- 7200 RPM
- 4.2ms latency (?)
- 100MB/Sec transfer speed
- 32MB cache

Properties of a Hard Magnetic Disk

- **Properties**
  - Independently addressable element: sector
  - 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**


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**
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

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

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%

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 = Queueing 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 \text{ RPM}/60\text{s/M})] \times 1000 \text{ ms/s} + [8192 \text{ bytes}/(4\times10^6 \text{ bytes/s})] \times 1000 \text{ ms/s} + 2 \text{ 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 × 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

A Little Queuing Theory: Mean Wait Time

- Parameters that describe our system:
  - \( \lambda \): mean number of arriving customers/second
  - \( T_{ser} \): mean time to service a customer ("m1")
  - \( C \): squared coefficient of variance = \( \sigma^2/m1^2 \)
  - \( \mu \): service rate = \( 1/T_{ser} \)
  - \( u \): server utilization (0 ≤ u ≤ 1): \( u = \lambda/\mu = \lambda \times T_{ser} \)

- Parameters we wish to compute:
  - \( T_q \): Time spent in queue
  - \( L_q \): Length of queue = \( \lambda \times T_q \) (by Little’s law)

- Basic Approach:
  - Customers before us must finish; mean time = \( L_q \times T_{ser} \)
  - If something at server, takes \( m1(z) \) to complete on avg
    - Chance server busy = \( u \) ⇒ mean time is \( u \times m1(z) \)

- Computation of wait time in queue (\( T_q \)):
  \( T_q = L_q \times T_{ser} + u \times m1(z) \)
Mean Residual Wait Time: $m_1(z)$

Total time for n services

| $T_1$ | $T_2$ | $T_3$ | ... | $T_n$ |

Random Arrival Point

- Imagine n samples
  - There are $n \times P(T_x)$ samples of size $T_x$
  - Total space of samples of size $T_x$: $T_x \times n \times P(T_x) = n \times T_x P(T_x)$
  - Total time for n services: $\sum x \times n \times T_x P(T_x) = n \times T_{ser}$
  - Chance arrive in service of length $T_x$: $n \times T_x P(T_x) = T_x P(T_x)$

- Avg remaining time if land in $T_x$: $\frac{1}{2} T_x$

- Finally: Average Residual Time $m_1(z)$:

$$\sum x \left( \frac{1}{2} T_x P(T_x) \right) = \frac{1}{2} T_{ser} \left( \frac{E(T_x^2)}{T_{ser}^2} \right) = \frac{1}{2} T_{ser} \left( \frac{\sigma^2 + T_{ser}^2}{T_{ser}^2} \right) = \frac{1}{2} T_{ser} \left( 1 + C \right)$$

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, $u = \frac{\lambda}{T_{ser}}$
  - 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:

  - $\bar{\lambda}$ (avg # arriving customers/s) = 10/s
  - $T_{ser}$ (avg time to service customer) = 20 ms
  - $T_q$ (avg time/customer in queue) = $T_{ser} \times u / (1 - u)$
  - $L_q$ (avg length of queue) = $\lambda \times T_{ser} = 10 / 0.02 = 0.2$ s
  - $T_{sys}$ (avg time/customer in system) = $T_q + T_{ser} = 25$ ms

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

- Computation of wait time in queue ($T_q$):

$$T_q = L_q \times T_{ser} + u \times m_1(z)$$

- Defn of utilization ($u$):

$$u = \frac{\lambda}{T_{ser}}$$

- Notice that as $u \to 1$, $T_q \to \infty$!

- 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: $T_q = T_{ser} \times \frac{1}{2}(1+C) \times u / (1 - u)$

- Memoryless service distribution ($C = 1$):
  - Called M/M/1 queue: $T_q = T_{ser} \times u / (1 - u)$

Conclusion

- ECC: add redundancy to correct for errors
  - $(n,k,d) \Rightarrow n$ 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: $W = \frac{1}{2} (1 + C) \frac{\bar{x}u}{1 - u}$ for ($c=1$): $W = \frac{\bar{x}u}{1 - u}$

4/6/2009 cs252-S09, Lecture 18

4/6/2009 cs252-S09, Lecture 18

4/26/2010 cs252-S10, Lecture 24

43

44