Distributed Dependable Systems

4. Erasure Coding

Problem Statement

- I have data to save
  - E.g., 100GB
- I can store data on N different servers
  - Say I pay storage per GB
- I want to be able to recover my data even if M servers fail and lose my data
- How to minimize the cost?

Trivial Solution: Replication

- To safeguard myself against M server failures, I put a copy of all my data on M+1 servers
- Of course, I'm safe if M servers die
- **Redundancy**—i.e., the ratio between amount of data I store and the amount of original data is M+1
  - E.g., for M=2 and 100GB of data, I need 300GB
  - Redundancy: 3

N=3, M=1: Parity

- I split my data in 2 **blocks** B_0 and B_1
  - E.g., N=2 blocks of 50GB each
- I create a **parity** redundant block B_R
  - $B_R = B_0 \text{ XOR } B_1$
- If I lose one of the blocks B_i, I can recover it as
  - $B_i = B_R \text{ XOR } B_{1-i}$
  - This is because x XOR (x XOR y) = y
- Redundancy: 1.5
  - E.g., for 100GB, I need 150GB storage
  - Only 100GB to recover all the original data

M=1, Any N

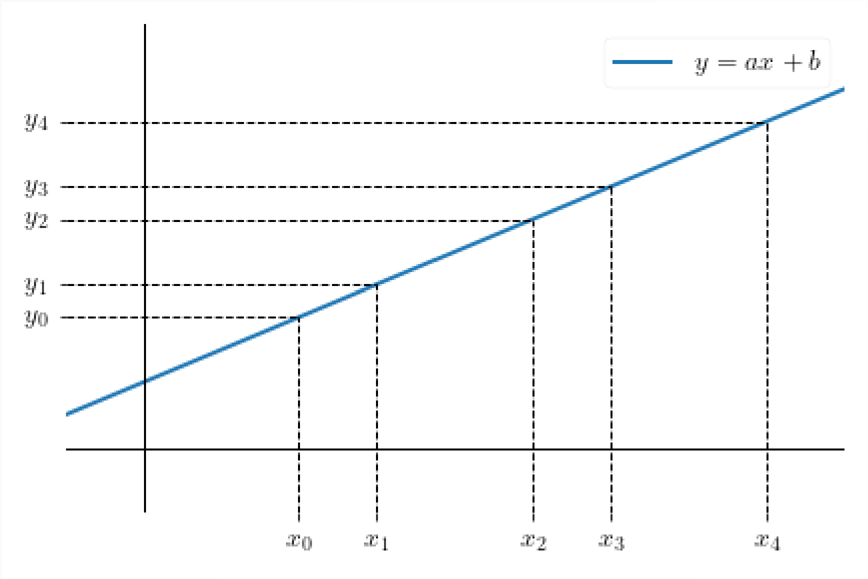
- I split my data in K=N-1 **blocks** of the same size
  - E.g., N=6, 100GB: 5 blocks B_0, …, B_{K-1} of 20GB each
- I create a **parity** redundant block B_R
  - $B_R = B_1$ XOR B_2 XOR … XOR B_N
- If I lose one of the blocks B_i, I can recover it as
  - $B_i = B_1$ XOR B_2 … XOR B_{i-1} XOR B_{i+1}… XOR B_{K-1} XOR B_R
  - This is because x XOR (x XOR y) = y
  - Here y=B_1 XOR B_2 … XOR B_{i-1} XOR B_{i+1}… XOR B_{K-1} XOR B_R
- Redundancy: N/K=N/(N-1)
  - Say N=6, 100GB: redundancy 6/5=1.2, I need 120GB
  - Again, only 100GB to recover all original data

Erasure Coding Magic: Any N & M

- I **encode** my data in N blocks, each of size 1/Kth of the original data, where K=N-M
  - E.g., M=2, N=6: 6 blocks of size 25GB
- I can decode **any** K of those blocks to recover my original data
  - E.g., any 4 of the N=6 blocks in the example
  - Once again, I just need any blocks totaling 100GB to recover my original data
- Redundancy: N/(N-M)
  - 1.5 in the example, 150GB total

A Day Trip Into Erasure Coding

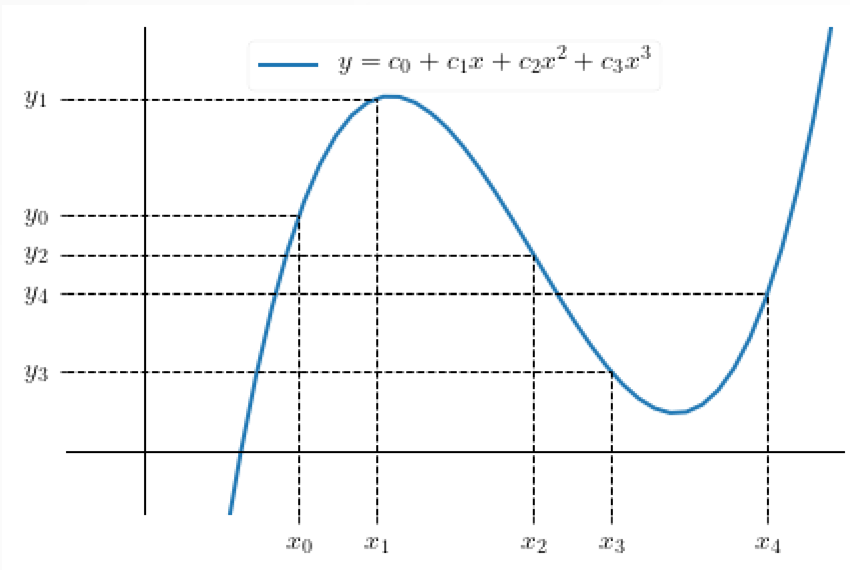
Any N, M=N-2: Linear Oversampling



- A single straight line connects any two distinct points
- Hence, we can compute *a* and *b* with any two (x_i, y_i) pairs
  - If values of x_i are predetermined, no need to send those

- Message: *a, b*
  - *e.g., a=3, b=2*
- Encoded message: $y_0=f(x_0)$, $y_1=f(x_1)$, …, $y_{n-1}=f(x_{n-1})$
  - With $x_i=i$: (2, 5, 7, …)
- With any two distinct (x_i, y_i) pairs, a system of two linear equations and 2 variables:
  - E.g., With $y_2=8$ and $y_5=17$:
    - $ax_2+b=8 \rightarrow 2a+b = 8$
    - $ax_5+b=17 \rightarrow 5a+b = 17$
  - From here, it's trivial to compute the message
- If message and all x_i are all integers, all y_i will be too: **no need to handle non-integer math**!

Any N&M: Polynomial Oversampling



- Any $K=N-M$ distinct points identify a single polynomial of degree $K-1$
- Hence, we can find the message with any K (x_i,y_i) pairs
  - Again, we can agree beforehand what all x_i values are
- Could sound familiar if you already studied **secret sharing** in cryptography

- Message: $c_0, \ldots, c_{K-1}$
- Encoded message: $f(x_0), f(x_1), \ldots, f(x_{N-1})$
- With any n distinct (x_i,y_i) pairs, a system of n **linear** equations and n variables
  - $c_0+c_1x_i+c_2x_i^2+\ldots+c_{k-1}x_i^{k-1}=y_i$
  - The nonlinear part **disappears** because all x_i values are known

Polynomial Oversampling: Example

- $N=7, M=3, K=4$
- Sampling points $x_0, \ldots, x_6 = 0, \ldots, 6$
- Message: $(c_0, \ldots, c_3)$; $f(x)=c_0+c_1x+c_2x^2+c_3x^3$
- Suppose we lose the other values and remain with $y_0=f(x_0)=f(0)=4$, $y_2=40$, $y_3=100$, $y_5=364$. We get the equations
  - $c_0=4$
  - $c_0+2c_1+4c_2+8c_3=40$
  - $c_0+3c_1+9c_2+27c_3=100$
  - $c_0+5c_1+25c_2+125c_3=364$
- We can obtain the original message by solving the equations (e.g., by substitution)
  - **Please do as an exercise**
  - Solution: $(c_0, \ldots, c_3) = (4, 2, 4, 2)$
  - Function $f(x)=4+2x+4x^2+2x^3$
  - Encoded message $y_0, \ldots, y_6 = 4, 12, 40, 100, 204, 364, 592$
- **Problem**: numbers grow in size! If we encode them as bits, numbers in the encoded message will be **bigger** than those in the original one
- Is there a magic way to make sure numbers don't become bigger as we multiply them by powers of x?
  - Yes there is! They're called ***finite fields*** (or Galois fields)

It's OK If You Don't Remember Fields

- Informally: a field is a set in which addition, substraction, multiplication and division are defined and behave "*as in*" real and rational numbers
- Disclaimer: in this and the following slides, **bold red** symbols **+**, **-**, *****, **/**, **0**, **1** and $^{\mathbf{-1}}$ refer to operation on the field, and not on the numbers we're used to
- Properties:
  - **Associativity**
    - *(a**+**b)**+**c=a**+**(b**+**c)* and *(a*****b)*****c=a*****(b*****c)*
  - **Commutativity**
    - *a**+**b=b**+**a* and *a*****b=b*****a*
  - **Additive & Multiplicative identity**
    - a**+0**=a, a***1**=a
  - **Distributivity**
    - a*****(b**+**c)=(a*****b)**+**(a*****c)
  - **Additive and multiplicative inverses**
    - a**+**(**-**a)=**0**, a*****(a$^{\mathbf{-1}}$)=**1** *(the multiplicative inverse is not defined for **0**)*
- **It's useful in our case because we can solve our linear equations** in a field
  - If you think about it, to solve them you do substitutions and or add/multiply/divide the same number from both sides of an equation

Finite Fields (Galois Fields)

- Fields that have **just a finite number of elements**
- Discovered (invented?) by Évariste Galois (1811-1832)
- They're cool, because we can give a number to each element of the field, and encode those numbers using $log_2(n)$ bits for a field of size n, and do everything as before!

Integers Modulo A Prime Number Are A Finite Field

- Elements of the set: $0, 1, \ldots, p\text{-}1$
- Obvious definition of 0, 1, multiplication, addition and additive inverse
  - $\mathbf{0}=0$, $\mathbf{1}=1$, $\mathbf{-}a=-a \bmod p$
  - $a\mathbf{+}b=a+b \bmod p$
  - $a\mathbf{*}b=ab \bmod p$
  - ***What about the multiplicative inverse?*** We'll see later
- Most properties are easy to prove (please try at home)… But what about the multiplicative inverse?

Multiplication Table Modulo *m*

	1	**2**	**3**	**4**	**5**	**6**
1	**1**	2	3	4	5	6
2	2	4	6	**1**	3	5
3	3	6	2	5	**1**	4
4	4	**1**	5	2	6	3
5	5	3	**1**	6	4	2
6	6	5	4	3	2	**1**

- $m=7$ (***prime***)
- There's exactly one 1 in every row and column
- In the "modulo 7" field, $5^{-1}=3$ and $2^{-1}=4$

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	**1**	2	3	4	5	6	7	8
2	2	4	6	8	**1**	3	5	7
3	3	6	0	3	6	0	3	6
4	4	8	3	7	2	6	**1**	5
5	5	**1**	6	2	7	3	8	4
6	6	3	0	6	3	0	6	3
7	7	5	3	**1**	8	6	4	2
8	8	7	6	5	4	3	2	**1**

- $m=9$, non-prime
- Not every value has an inverse
- Not a field

Proof: There's Exactly One Multiplicative Inverse In Modulo p

- Consider $a*b=ab \bmod p$, *with a, b in [0, p-1]*
  - It can be zero only if one of a or b is **0**, because:
    - p is a prime, so ab mod p=0 only if one of a and b is a multiple of *p*
    - but they can't be multiples of p because they're smaller than *p*.
- Consider *a* in *[1,p-1]* and $a*i$ for all *i* in *[1,p-1]*
  - For all *i* in *[1,p-1]*, $a*i \neq 0$, neither *a* nor *i* are **0**
  - For all $i \neq j$ *in [1,p-1]*, $a*i \neq a*j$, because:
    - $a*i=a*j$ would mean $a(i-j)=0$ (distributive property)
    - a $\neq 0$ by hypothesis, i-j $\neq$0 because $i \neq j$
  - Conclusion: all the *p-1* $a*i$ values are different, and they can assume only the *p-1* values in *[1,p-1]*
    - Each value will be represented exactly once, hence there will be a single unique value of *i* such that $a*i=1$
    - Hence, that will be the multiplicative inverse a^{-1}

Our Example, Modulo p

- $N=7$, $M=3$, $K=4$ *as before), $\boldsymbol{p=7}$ (we need $p \geq N$)
- Function $f(x)=c_0+(c_1*x)+(c_2*x^2)+(c_3*x^3)=c_0+c_1x+c_2x^2+c_3x^3$ ***mod 7***
- Sampling points $x_0, \ldots, x_6 = 0, \ldots, 6$
  - **we're using all of them now, to add more we need to change *p***
- Suppose we lose the others and remain with $y_0=f(0)=3$, $y_2=1$, $y_3=5$, $y_5=3$. We get the equations
  - $c_0=3$
  - $c_0+2*c_1+4*c_2+c_3=1$
  - $c_0+3*c_1+2*c_2+6*c_3=5$
  - $c_0+5*c_1+4*c_2+6*c_3=3$
- Exercise: **solve this!** It's not so hard, you convert every number with its value modulo 7
  - E.g. -2 becomes 5 and 22 becomes 1
  - Inverses: $1^{-1}=1$, $2^{-1}=4$, $3^{-1}=5$, $4^{-1}=2$, $5^{-1}=3$, $6^{-1}=6$
  - To divide by x, you multiply by x^{-1} left and right
- **Solution**: $(c_0, \ldots, c_3) = (3, 1, 5, 4)$, $f(x)=3+x+5(x^2 \bmod 7)+4(x^3 \bmod 7)$
  - Encoded message $y_0, \ldots, y_6 = 3, 6, 1, 5, 0, 3, 3$
- Try making others up! It's easy, you convert every number with its value modulo 7

We Stop Here With Theory

- Almost-practical usage:
  - N is the number of machines that will store your data
  - M is the number of failures you want to tolerate
  - Choose p not smaller than N
  - Divide your data in K=N-M blocks of the same size
    - (find a way, e.g. padding, to make N a multiple of K)
  - Encode it as a series of values smaller than p
  - All elements of the first (original) block will be c_0 coefficients, second block c_1 and so on
  - Encode and put all the y_0 coefficients in the first encoded block, the y_1 coefficients in the second block, and so on

There's A Lot More In Coding Theory

- There are finite fields having p^m elements, any $m \geq 1$
  - But they're harder to explain
  - Of course, computer people in practice use 2^m
- You can use coding to do **error correction** in addition to handle erasures
- They're implemented in hardware
- Found everywhere: telecommunications, QR codes, even CD readers from the 90's!
- If you want to play with them, look for a Reed-Solomon library in your favorite programming language

More Coding Magic

- Approaches that “waste” a bit of space compared to the “perfect” result but give you great properties
  - I.e., to recover 100 GB of data you’ll need a bit more than 100 GB
- **Fountain codes**: you don’t need to choose N to start with
  - Generate as many redundant blocks as you want, as a stream
  - You need a bit more than the amount of the original data to reconstruct it
  - Current standard: RaptorQ, IEEE RFC 6330
- **Regenerating codes** (Dimakis et al. 2010): if you lose one or a few blocks you don’t need the original plaintext to recreate them—just download a few encoded blocks and work from them

Let's Simulate

- We can return to discrete event simulation and simulate a **backup application**
  - Can we keep data safe for 100 years?
  - No need to actually do the encoding here, we can claim success if $K=N-M$ blocks are successfully recovered after a disk crash
- Consider a home machine that has, say, 100 GB to backup and can store data on $N=10$ servers, 500 kBps uplink speed and 2 MBps downlink
  - This machine goes online/offline. For a machine that stays online on average x continuous hours every day, we can use an exponential distribution having average x hours (x*3600 s) for the online intervals, and $24-x$ hours for the offline intervals
- Data is split in N blocks of size 100/K GB, and uploaded sequentially to the N servers
- Each server experiences a disk crash after an exponentially distributed amount of time with average y days
  - Stored data is lost
  - The home machine will re-upload the lost block
- After a time chosen exponentially with average z days, the machine's disk will crash, and it will try to download K blocks from the servers to recover its backup
  - Play around with different values of K ($K=1$: replication, $K=N$: no redundancy), and compare costs (total amount of data stored on server) and probability of recovering backup

Questions

- What is the interplay between the parameters? Which are the most important ones?
- If you were to design the system, how would you tune it to get the best reliability?
- What if we drop the assumption of storing a single block per server? Does the system become better?
- This will be a second piece of work we'll ask at the exam, like the one on scheduling