Cloud Storage using Erasure Code

Distributed Storage

- Striping/Sharding
  - How to store objects in distributed storage
    - A lot of data objects (or files) in many storage nodes
    - A large data object in many storage nodes ⇒ Striping
      - We have discussed how to partition, placing, maintaining layout
  - What are the problems with striping?
    - One node failure ⇒ data loss
      - One node failure: f ⇒ At least one of N node failure: $1-(1-f)^N$
  - Solutions
    - Striping + replication ⇒ High space consumption
    - (N,K) erasure code
      - Divide data into N shares, reconstruct data from K shares

Erasure Code

❖ (N,K) erasure code

- Code with original data
  - RAID is also a type of erasure code with original data

$$\begin{array}{ccccccc} d_1 & d_2 & d_3 & d_4 & r_1 & r_2 & r_3 \\ & d_2 & & d_4 & r_1 & r_2 & \end{array} \Rightarrow \begin{array}{ccccccc} d_1 & d_2 & d_3 & d_4 & r_1 & r_2 & r_3 \end{array}$$

- Code without original data
  - May be for security, e.g. secret sharing
  - E.g., N = 10, K = 4, take any 4, original data can be reconstructed

$$\begin{array}{cccccccccc} c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} \\ & & c_3 & c_4 & & c_6 & c_7 & & & \end{array} \Rightarrow \begin{array}{cccc} d_1 & d_2 & d_3 & d_4 \end{array}$$

Erasure Code

❖ Different focuses for communication and storage

- Most erasure codes are originally designed for reliable data transmission
- In communication
  - Error checking and recovery are both important
- In storage
  - Consider storage node failures, but failure detection is not the main concern, only need to consider recovery from failures
    - Each piece of data is stored at a different storage ⇒ Failed node is clearly known
    - Many disks have their own integrity coding schemes to detect errors

Erasure Code, Replication, Striping

- What are to be compared
  - Redundancy level
    - How much more redundant space is needed to achieve the same level of fault tolerance
  - Parallel transmission
    - When accessing a certain data by one client, different parts of the data can be transmitted from different nodes
    - Reduce the communication latency
  - Parallel accesses
    - When accessing the same data by different clients, concurrent client accesses can be parallelized
    - Reduce the potential waiting time

Erasure Code, Replication, Striping

- Redundancy level
  - Replication
    - Consider 3 replicas: with 2 failures, there is still one good replica
    - Storage space is tripled, redundancy is 200%
  - (N, K) erasure code
    - 4 data nodes, 2 redundancy nodes
    - Can also tolerate 2 failures
    - Only 50% redundancy
    - Any disadvantage?
      - Recovery time will be higher
  - Striping + replication
    - Same redundancy as replication
      - (striping alone has no redundancy, no fault tolerance)

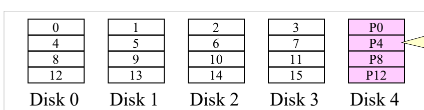
d_1	d_2	d_3	d_4	r_1	r_2

Erasure Code, Replication, Striping

- Parallel accesses and communication cost
  - Striping + replication $d_1\ d_2\ d_3\ d_4\ d_1\ d_2\ d_3\ d_4\ d_1\ d_2\ d_3\ d_4$
    - Cannot access arbitrarily
    - Has to get one block from each specific node group
  - (N,K) erasure code $d_1\ d_2\ d_3\ d_4\ r_1\ r_2\ r_3\ r_4\ r_5\ r_6\ r_7\ r_8$
    - Assume MDS: Maximum distance separable code
    - Data can be reconstructed from any K (out of N) shares
    - Increase N to increase the level of parallel accesses
    - Access the nearest K nodes to reduce the communication cost
    - The data encoding and reconstruction cost is a concern

RAID

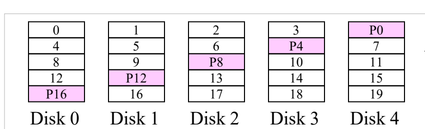
- RAID 4



Read: do not access parity
Write: need to read/write parity
⇒ Unbalanced disk load

0	0	0	0	0
1	0	0	0	1
0	0	1	1	0
1	1	0	0	0

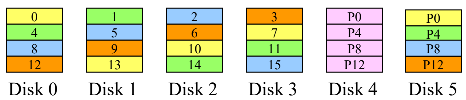
- RAID 5



Distribute parity disk
⇒ Balanced disk load for both read and write operations

RAID

- RAID 6
  - 2 parity disks
  - How to compute the second parity? Diagonal
    - Consider K data disks and K data size (w = K)



0	0	0	0	0	0
1	0	0	0	1	0
0	0	1	1	0	0
1	1	0	0	0	1

  - But recovery may be a problem!

RAID

- Recovery problem in simple 2-parity solution
  - Consider two disk failures

0	?	0	?	0	0
0	?	0	?	0	0
0	?	0	?	0	0
0	?	0	?	0	0

  - Multiple solutions

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

0	1	0	1	0	0
0	1	0	1	0	0
0	1	0	1	0	0
0	1	0	1	0	0

EvenOdd Coding

- EvenOdd coding
  - Consider K disks, K–1 block size
    - K is a prime

0	1	1	0	0	0	1
1	1	0	0	1	1	0
0	1	1	1	0	1	0
0	0	0	1	1	0	0
0	0	0	0	0	0	0

Pseudo row, every bit should be 0
(5,7) is diagonal parity for gray
Need to make this bit (5,7) = 0
⇒ Let this be the adjust bit A
⇒ For this example, A = 1
⇒ Use A to adjust all the diagonal parity bits

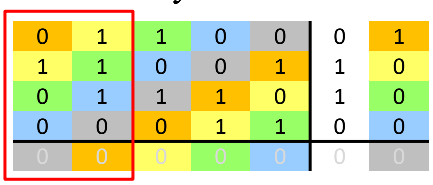
  - How to compute A from parities?
    - XOR (horizontal parities)
      = XOR (all data bits, K*(K–1) bits)
      = XOR (diagonal parities) + A
    - In this example: XOR(0110) = XOR(1000) + A ⇒ A = 1

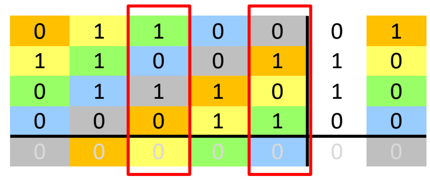
EvenOdd Coding

- Recovery
  - How to recover from 1 disk failure?
    - Data disk failure: recover from one of the parity disks
    - Parity disk failure: recompute the lost parity
  - How to recover from 2 disk failures?
    - 1 data disk + diagonal parity disk
      - First recover the data disk from horizontal parity disk
      - Then recompute the diagonal parity disk
    - 1 data disk + horizontal parity disk
    - 2 data disks
  - Why do we need to adjust the diagonal parity???

EvenOdd Coding

❖ Recovery from 2 data disk failures

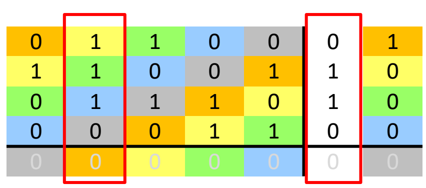




- For any two data disk failures, there are always two diagonals that only lost one bit ⇒ derive those lost bits first
  - Disks 1 and 2 fail: orange bit M[1,1] can be computed first
  - Disks 3 and 5 fail: blue bit M[2,3] can be computed first
- After the first recovery, one row can be recovered fully, then zig-zag to recover the rest

EvenOdd Coding

❖ Recovery from 1 data disk+ horizontal parity disk failures



- With any 1 data disk failure, there is always one diagonal without data loss ⇒ Derive the adjust bit from that diagonal

Even-Odd Coding

- Why N has to be prime?
  - (N: # data disks, e.g., 4, 6)
  - By counter example

	1	2	3	4		
1		0		0	0	0
2	0	0	0	0	0	0
3		0		0	0	0
0	0	0	0	0	0	0

	1	2	3	4	5	6		
1		0		0	0	0	0	0
2	0	1	0	1	0	0	0	1
3		0		0	0	0	0	0
4	0	1	0	1	0	0	0	0
5		0		0	0	0	0	1
0	0	0	0	0	0	0	0	0

When #data-disk is even, (N = 4/6):
Choose disks 1,3 as failed (gray, green)
⇒ Gray on disk 3 is at row (N–(3–1) = 2/4)
⇒ Start from row 2 on disk 1, then disk 3,
next row is always "2" rows after
i.e., row (s*2) % N, s = 1..N–1
⇒ Always get to gray before finish ⇒ Stuck
(because gray lost only one bit,
after recovering the lost bit,
cannot proceed on diagonal any more)

When #data-disk is multiple of 3 (e.g., 9/15)
Choose disks 1,4 as failed (gray, green)
⇒ Gray on disk 4 is at (N+1–4 = 6/12)
⇒ Start from position 3, jump 3
⇒ Always get to gray before finish ⇒ Stuck

RAID-DP

- RAID double parity
  - Used in RAID-6
  - Consider K disk and K data bits from each disk
    - K+1 is a prime

0	1	1	0	0	1
1	1	0	0	0	0
0	1	1	1	1	1
0	0	0	1	1	1

  - Horizontal parity
  - Diagonal parity
    - The horizontal parity is also used to compute the diagonal parity
    - Gray diagonal has no diagonal parity

RAID-DP

	1	2	3	4	5	
1	0	1	1	0	0	1
2	1	1	0	0	0	0
3	0	1	1	1	1	1
4	0	0	0	1	1	1
0	0	0	0	0	0	0

1 disk failure or
1 data disk + D-parity disk
⇒ no problem

- Any two disk failures, not the diagonal parity
  - Horizontal parity disk is the same as data disk
    - They are horizontal parities for each other
  - If disk 1 and another disk failed (e.g., 3)
    - Always can find a non-gray diagonal with only 1 lost bit
      - The other diagonal is gray, which also lost only one bit
    - Blue: (2,1) → (2,3) → (4,1) → (4,3) → (1,1) → (1,3) → (3,1) → (3,3) → done

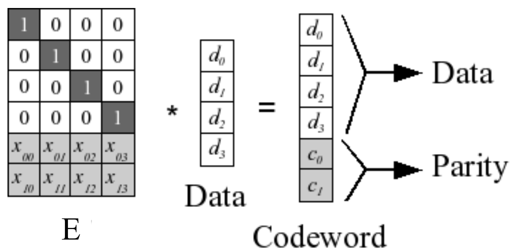
RAID-DP

	1	2	3	4	5	
1	0	1	1	0	0	1
2	1	1	0	0	0	0
3	0	1	1	1	1	1
4	0	0	0	1	1	1
0	0	0	0	0	0	0

- Any two disk failures, not the diagonal parity
  - If failed does not include disk 1 (e.g., 2 and 5)
    - Always can find 2 non-gray diagonals with only 1 lost bit
    - Orange: (3,2) → (3,5) → (1,2) → (1,5) → stuck
    - Green: (2,5) → (2,2) → (4,5) → (4,2) → stuck, but done

Reed Solomon Code

- Generation matrix
  - Identify matrix I_n
    - With dimension n*n
  - Additional rows: Redundancy matrix
    - Vandermonde and Cauchy constructions Assure inversibility
  - In encoding, no need to compute first n rows
    - They are the original data (because of the identify matrix)



Reed Solomon Code

- Vandermonde matrix
  - E = $\begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^{n-1} \\ 1 & \alpha_3 & \alpha_3^2 & \dots & \alpha_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_m & \alpha_m^2 & \dots & \alpha_m^{n-1} \end{bmatrix}$
- Cauchy matrix

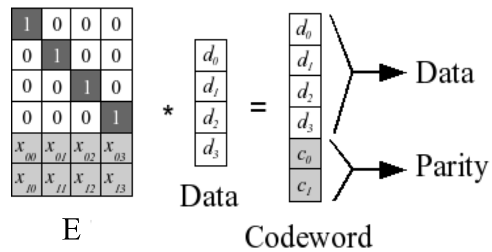
$$E^T = a_{ij} = \frac{1}{x_i - y_j}; \quad x_i - y_j \neq 0, \quad 1 \le i \le m, \quad 1 \le j \le n$$

  - Special case of Cauchy matrix: Hilbert matrix
    - $x_i - y_j = i + j - 1$
  - Every sub-matrix of a Cauchy matrix is a Cauchy matrix

Size of the Code

- All (N,K) coding can be expressed using encoding and decoding matrices
  - What's the size of c_i?
    - Assume regular *, +
  - d_i is 8 bits
    - c_i will be 18 bits
  - d_i is 32 bits
    - c_i will be 66 bits
  - d_i is b bits

    $\Rightarrow c_i$ will be $b*2 + \log n$ bits



Size of Code

- Code size should be the same as data size
    - Have a longer code than the data size is waste of space
  - Field
    - Closure for addition and multiplication
      - If $x, y \in F$, then $x + y, x * y \in F$
    - Existence of additive & multiplicative inverse in F
      - If $x \in F$, then $\exists\, y \in F$, s.t. $x + y = 0$ (or $x * y = 1$)
    - Coding computation on a field is reversible
  - Choice of field for RS computation
    - Integer is a field, but data size has problem
    - Prime field: a + b $\Rightarrow$ (a + b) % p; a * b $\Rightarrow$ (a * b) % p
      - Prime will not be 2^x $\Rightarrow$ One bit longer in code size than in data size
    - Galois' Field: Final choice in coding

Galois' Field Arithmetic

- Galois field with 2^N elements
  - Consider an irreducible polynomial P
    - $P = a_0 x^N + a_1 x^{N-1} + \ldots + a_{N-1} x + a_N$
      - a_i, for all i, is in {0, 1}
    - mod P is a field
      - (Similar to the concept of mod a prime number)
  - Computation
    - If p_1 and p_2 are in P, then $p_1 + p_2$ and $p_1 * p_2$ are in P
      - $(p_1 + p_2)$ mod P, $(p_1 * p_2)$ mod P
  - Use Galois field in RS computation
    - Map polynomial to the binary numbers
    - + becomes XOR and * can be done by table lookup

Galois' Field Arithmetic

- Galois field with 4 elements
  - Consider an irreducible polynomial $x^2 + x + 1$
    - $x^2 + x + 1 = 0 \Rightarrow x^2 = -(x + 1) = x + 1$
      - In Z_2, $-1 = -1 + 2 = 1$
    - $(1+x)^2 = 1 + 2x + x^2 = x$
    - $(1+x) * x = x + x^2 = -1 = 1 \Rightarrow x$ and $(1+x)$ inverse to each other
      - In Z_2, $2x = 0$
    - Map to the 2-bit binary numbers
      - $0 + 0x \Rightarrow 00$; $1 + 0x \Rightarrow 10$; $0 + 1x \Rightarrow 01$; $1 + x \Rightarrow 11$

Erasure Code in Storage Systems

- Erasure code is used quite often in cloud storage
  - Microsoft Azure
  - GFS II
  - Facebook HDFS
  - Mostly used for redundancy
    - Focus on improving recovery speed, i.e., when a share is lost, how to rebuild it efficiently
    - If we reconstruct the data, the share can be recomputed, but can we do better?
    - All the systems above use an improved method, focusing on recovery speed

Erasure Code in GFS II

- Plain RS 6+3 code in Google GFS II
  - (N,K) = (9, 3)

d_0	d_1	d_2	d_3	d_4	d_5	p_1	p_2	p_3

Erasure Code in WAS

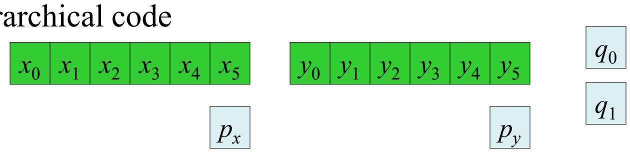
- Plain RS 12+4 code

$d_0\ d_1\ d_2\ d_3\ d_4\ d_5\ d_6\ d_7\ d_8\ d_9\ d_{11}\ d_{12}$ $\quad$ $p_1\ p_2\ p_3\ p_4$

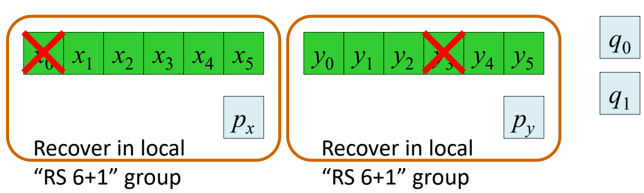
- Reliability, redundancy, repair cost
  - 6+3
    - Recovery ratio = 3/9 = 0.33
    - Redundancy = 3/6 = 0.5
    - Repair cost = 6 shares transferred
  - 12+4
    - Recovery ratio = 4/16 = 0.25
    - Redundancy = 4/12 = .33
    - Repair cost = 12 shares transferred $\Rightarrow$ very expensive!!!

Erasure Code in WAS

- Azure's erasure code
  - Hierarchical code

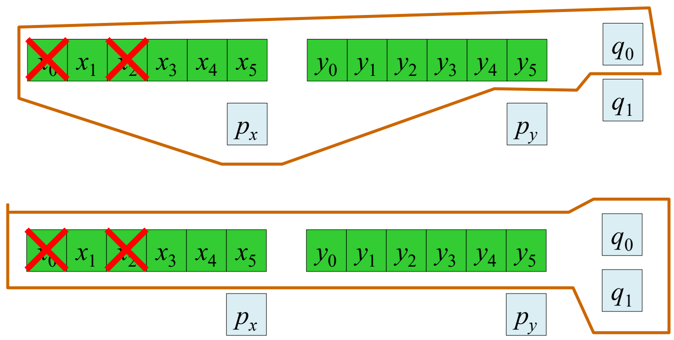


  - Handling 1failure or 2 failures in different groups
    - Require 6 share transfers



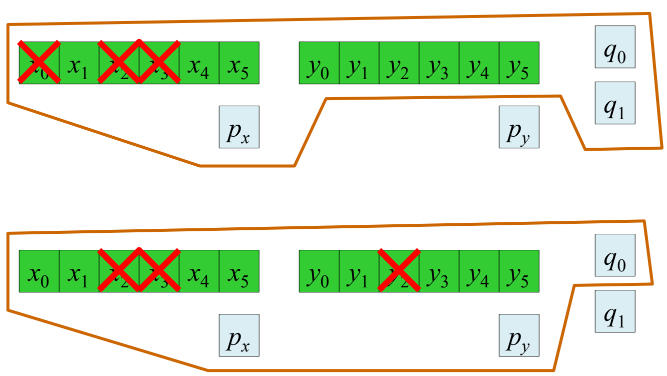
Erasure Code in WAS

- Azure's erasure code
  - Handling 2 failures in the same group
    - Require 12 share transfers in either way



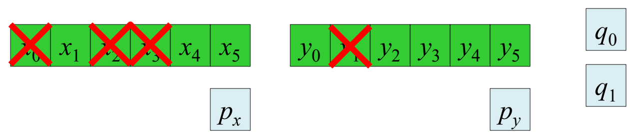
Erasure Code in WAS

- Azure's erasure code
  - Handling 3 failures

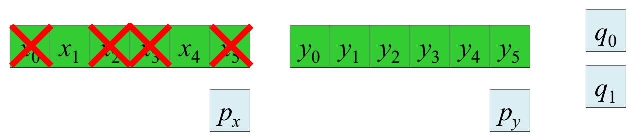


Erasure Code in WAS

- Azure's erasure code
  - May handle some of the 4 failure cases
    - As long as failures occur in both groups, all four redundant shares are useful

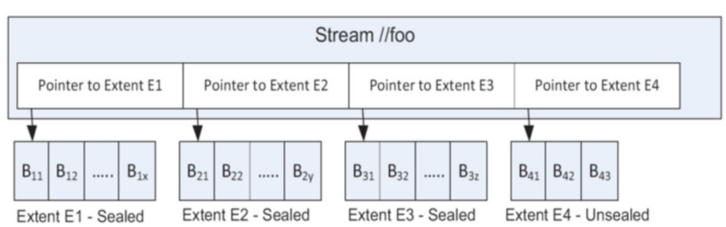


    - If failures are all in one group, (p_y cannot be made use of in the example) ⇒ cannot recover, but chance of this is relatively low



Windows Azure Storage

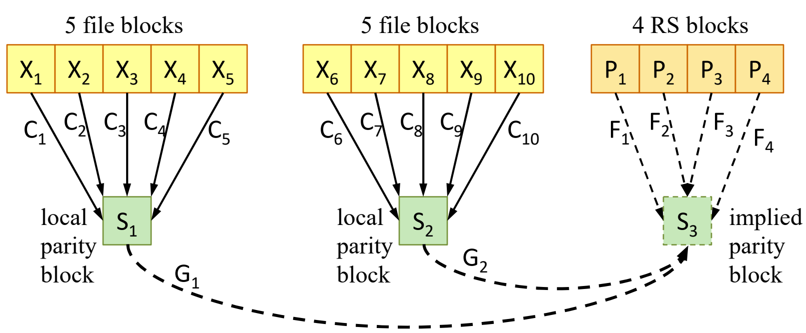
- File structure
  - File is considered as a stream, and is "append only"
  - Extent: unit of placement/replication
  - Block: regular file blocks, each extent has a set of blocks
  - Stream means append only ⇒
    - Only one extent open for append, the rest are "sealed"
    - Replication for unsealed, erasure coding for "cold" sealed



Erasure Code in Facebook HDFS-Xorbas

- ❖ Locally repairable code (LRC)
  - A type of regenerate code
  - A type of hierarchical regenerate code
    - A simplified version
    - Give a new name due to the naming game
  - Facebook HDFS Xorbas

Erasure Code in Facebook HDFS-Xorbas



- $S_1 = C_1X_1+C_2X_2+C_3X_3+C_4X_4+C_5X_5$
- $S_2 = C_6X_6+C_7X_7+C_8X_8+C_9X_9+C_{10}X_{10}$

- $S_3 = F_1P_1+F_2P_2+F_3P_3+F_4P_4$
  $= G_1S_1+G_2S_2$
- S_3 is implied, not actually stored

Erasure Code in Facebook HDFS-Xorbas

- Handling failures
  - If at most one failure in each group $\Rightarrow$ local repair
    - If P_1 fails, compute $S_3 = G_1S_1+G_2S_2$, and then derive P_1 from S_3
  - If more than one failure in a group $\Rightarrow$ require RS repair
  - No problem in handling any 6 failures
    - No problem with 4 failures
    - If all 5 blocks in a group fail, recover from 1 local + 4 RS
    - The 6th failure has to be in a different group
  - If only 3 RS blocks are used
    - Same as the case in Azure erasure code, may not be able to recover 5 failures with 5 redundant blocks

References

- References
  - RAID

    - A case for redundant arrays of inexpensive disks (RAID)
    - M. Blaum, J. Brady, J. Bruck and J. Menon, "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," IEEE Transactions on Computing, Vol. 44, No. 2, Feb. 1995, pp. 192-202.
    - Jay White, Chris Lueth, Jonathan Bell, "RAID-DP: NetApp Implementation of Double-Parity RAID for Data Protection" NetApp.com, March 2003.
    - C. Huang and L. Xu, "STAR: An efficient coding scheme for correcting triple storage node failures," Usenix Conference on File and Storage Technologies (FAST), December, 2005, pp. 197-210.

References

- References
  - Reed Solomon
    - I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," Journal of the Society for Industrial and Applied Mathematics, 8, 1960, pp. 300-304.
    - Vandermonde matrix: http://en.wikipedia.org/wiki/Vandermonde_matrix
    - Cauchy Reed-Solomon: http://planetmath.org/CauchyMatrix.html
  - Azure erasure storage
    - C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, S. Yekhanin, "Erasure coding in Windows Azure storage," USENIX ATC, June 2012
  - Facebook HDFS-Xorbas
    - M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A.G. Dimakis, R. Vadali, S. Chen, D. Borthakur. "XORing elephants: Novel erasure codes for big data," VLDB 2013, pp. 325-336