Erasure Codes for Systems

COS 518: Advanced Computer Systems
Lecture 14

Michael Freedman

Slides originally by Wyatt Lloyd

Things Fail, Let's Not Lose Data

- Replication
  - Store multiple copies of the data
  - Simple and very commonly used!
  - But, requires a lot of extra storage
- Erasure coding
  - Store extra information we can use to recover the data
  - Fault tolerance with less storage overhead

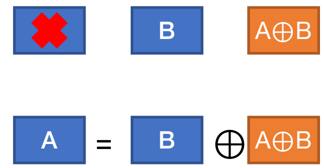
Erasure Codes vs Error Correcting Codes

- Error correcting code (ECC):
  - Protects against **errors** is data, i.e., silent corruptions
  - Bit flips can happen in memory -> use ECC memory
  - Bits can flip in network transmissions -> use ECCs
- Erasure code:
  - Data is **erased**, i.e., we know it's not there
  - Cheaper/easier than ECC
    - Special case of ECC
  - What we'll discuss today and use in practice
    - Protect against errors with checksums

Erasure Codes, a simple example w/ XOR

Erasure Codes, a simple example w/ XOR

Erasure Codes, a simple example w/ XOR



Reed-Solomon Codes (1960)

- N data blocks
- K coding blocks
- M = N+K total blocks

- Recover any block from any N other blocks!

- Tolerates up to K simultaneous failures

- Works for any N and K (within reason)

Reed-Solomon Code Notation

- N data blocks
- K coding blocks
- M = N+K total blocks

- RS(N,K)
  - (10,4): 10 data blocks, 4 coding blocks
    - f4 uses this, FB HDFS for data warehouse does too

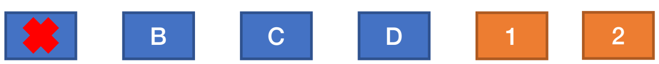
- Will also see (M, N) notation sometimes
  - (14,10): 14 total blocks, 10 data blocks, (4 coding blocks)

Reed-Solomon Codes, How They Work

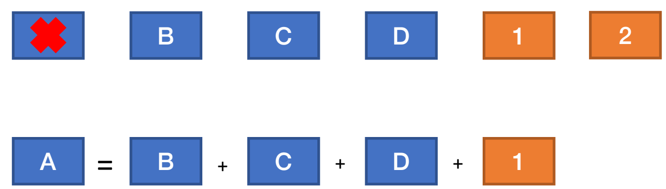
- Galois field arithmetic is the secret sauce
- Details aren't important for us ☺
- See "J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. Software—Practice & Experience 27(9):995–1012, September 1997."

Reed-Solomon (4,2) Example

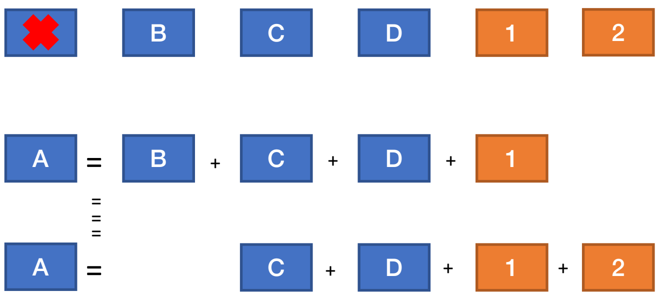
Reed-Solomon (4,2) Example



Reed-Solomon (4,2) Example



Reed-Solomon (4,2) Example



Erasure Codes Save Storage

- Tolerating 2 failures
  - 3x replication = ___ storage overhead

Erasure Codes Save Storage

- Tolerating 2 failures
  - 3x replication = 3x storage overhead
  - RS(4,2) = ___ storage overhead

Erasure Codes Save Storage

- Tolerating 2 failures
  - 3x replication = 3x storage overhead
  - RS(4,2) = (4+2)/4 = 1.5x storage overhead

Erasure Codes Save Storage

- Tolerating 2 failures
  - 3x replication = 3x storage overhead
  - RS(4,2) = (4+2)/4 = 1.5x storage overhead
- Tolerating 4 failures
  - 5x replication = 5x storage overhead
  - RS(10,4) = ___ storage overhead

Erasure Codes Save Storage

- Tolerating 2 failures
  - 3x replication = 3x storage overhead
  - RS(4,2) = (4+2)/4 = 1.5x storage overhead
- Tolerating 4 failures
  - 5x replication = 5x storage overhead
  - RS(10,4) = (10+4)/10 = 1.4x storage overhead
  - RS(100,4) = ___ storage overhead

Erasure Codes Save Storage

- Tolerating 2 failures
  - 3x replication = 3x storage overhead
  - RS(4,2) = (4+2)/4 = 1.5x storage overhead
- Tolerating 4 failures
  - 5x replication = 5x storage overhead
  - RS(10,4) = (10+4)/10 = 1.4x storage overhead
  - RS(100,4) = (100+4)/100 = 1.04x storage overhead

What's the Catch?

Catch 1: Encoding Overhead

- Replication:
  - Just copy the data
- Erasure coding:
  - Compute codes over N data blocks for each of the K coding blocks

Catch 2: Decoding Overhead

- Replication
  - Just read the data
- Erasure Coding

Catch 2: Decoding Overhead

- Replication
  - Just read the data
- Erasure Coding
  - Normal case is no failures -> just read the data!
  - If there are failures
    - Read N blocks from disks and over the network
    - Compute code over N blocks to reconstruct the failed block

Catch 3: Updating Overhead

- Replication:
  - Update the data in each copy
- Erasure coding
  - Update the data in the data block
  - And all of the coding blocks

Catch 3': Deleting Overhead

- Replication:
  - Delete the data in each copy
- Erasure coding
  - Delete the data in the data block
  - Update all of the coding blocks

Catch 4: Update Consistency

- Replication:
- Erasure coding

Catch 4: Update Consistency

- Replication:
  - Consensus protocol (Paxos!)
- Erasure coding
  - Need to consistently update all coding blocks with a data block
  - Need to consistently apply updates in a total order across all blocks
  - Need to ensure reads, including decoding, are consistent

Catch 5: Fewer Copies for Reading

- Replication
  - Read from **any** of the copies
- Erasure coding
  - Read from **the** data block
  - Or reconstruct the data on fly if there is a failure

Catch 6: Larger Min System Size

- Replication
  - Need K+1 disjoint places to store data
  - e.g., 3 disks for 3x replication
- Erasure coding
  - Need M=N+K disjoint places to store data
  - e.g., 14 disks for RS(10,4) replication

What's the Catch?

- Encoding overhead
- Decoding overhead
- Updating overhead
  - Deleting overhead
- Update consistency
- Fewer copies for serving reads
- Larger minimum system size

Different codes make different tradeoffs

- Encoding, decoding, and updating overheads
- Storage overheads
  - Best are "Maximum Distance Separable" or "MDS" codes where K extra blocks allows you to tolerate any K failures
- Configuration options
  - Some allow any (N,K), some restrict choices of N and K
- See "Erasure Codes for Storage Systems, A Brief Primer. James S. Plank. Usenix ;login: Dec 2013" for a good jumping off point
  - Also a good, accessible resource generally

Erasure Coding Big Picture

- Huge Positive
  - Fault tolerance with less storage overhead!
- Many drawbacks
  - Encoding overhead
  - Decoding overhead
  - Updating overhead
    - Deleting overhead
  - Update consistency
  - Fewer copies for serving reads
  - Larger minimum system size

Let's Improve Our New Hammer!

Erasure Coding Big Picture

- Huge Positive
  - Fault tolerance with less storage overhead!
- Many drawbacks
  - Encoding overhead
  - Decoding overhead
  - Updating overhead
    - Deleting overhead
  - Update consistency
  - Fewer copies for serving reads
  - Larger minimum system size

Immutable data

Erasure Coding Big Picture

- Huge Positive
  - Fault tolerance with less storage overhead!
- Many drawbacks
  - Encoding overhead
  - Decoding overhead
  - ~~Updating overhead~~
    - Deleting overhead
  - ~~Update consistency~~
  - Fewer copies for serving reads
  - Larger minimum system size

Immutable data

Erasure Coding Big Picture

- Huge Positive
  - Fault tolerance with less storage overhead!
- Many drawbacks
  - Encoding overhead
  - Decoding overhead
  - ~~Updating overhead~~
    - Deleting overhead
  - ~~Update consistency~~
  - Fewer copies for serving reads
  - Larger minimum system size

Immutable data

Storing lots of data
(when storage overhead
actually matters this is true)

Erasure Coding Big Picture

- Huge Positive
  - Fault tolerance with less storage overhead!
- Many drawbacks
  - Encoding overhead
  - Decoding overhead
  - ~~Updating overhead~~
    - Deleting overhead
  - ~~Update consistency~~
  - Fewer copies for serving reads
  - ~~Larger minimum system size~~

Immutable data

Storing lots of data (when storage overhead actually matters this is true)

Erasure Coding Big Picture

- Huge Positive
  - Fault tolerance with less storage overhead!
- Many drawbacks
  - Encoding overhead
  - Decoding overhead
  - ~~Updating overhead~~
    - Deleting overhead
  - ~~Update consistency~~
  - Fewer copies for serving reads
  - ~~Larger minimum system size~~

Data is stored for a long time after being written

Immutable data

Storing lots of data (when storage overhead actually matters this is true)

Erasure Coding Big Picture

- Huge Positive
  - Fault tolerance with less storage overhead!
- Many drawbacks
  - Encoding overhead
  - Decoding overhead
  - ~~Updating overhead~~
    - Deleting overhead
  - ~~Update consistency~~
  - Fewer copies for serving reads
  - ~~Larger minimum system size~~

Low read rate

Data is stored for a long time after being written

Immutable data

Storing lots of data (when storage overhead actually matters this is true)

Erasure Coding Big Picture

- Huge Positive
  - Fault tolerance with less storage overhead!
- Many drawbacks
  - Encoding overhead
  - Decoding overhead
  - ~~Updating overhead~~
    - Deleting overhead
  - ~~Update consistency~~
  - ~~Fewer copies for serving reads~~
  - ~~Larger minimum system size~~

Low read rate

Data is stored for a long time after being written

Immutable data

Storing lots of data (when storage overhead actually matters this is true)

f4: Facebook's Warm BLOB Storage System [OSDI '14]

Subramanian Muralidhar*, Wyatt Lloyd*φ, Sabyasachi Roy*, Cory Hill*, Ernest Lin*, Weiwen Liu*, Satadru Pan*, Shiva Shankar*, Viswanath Sivakumar*, Linpeng Tang*+, Sanjeev Kumar*

*Facebook Inc., φUniversity of Southern California, +Princeton University

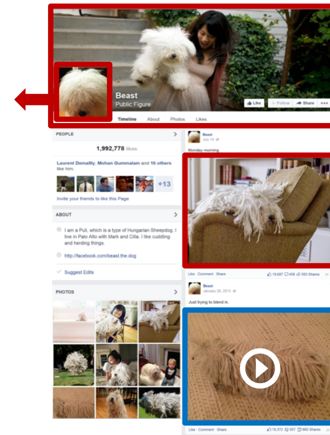
1

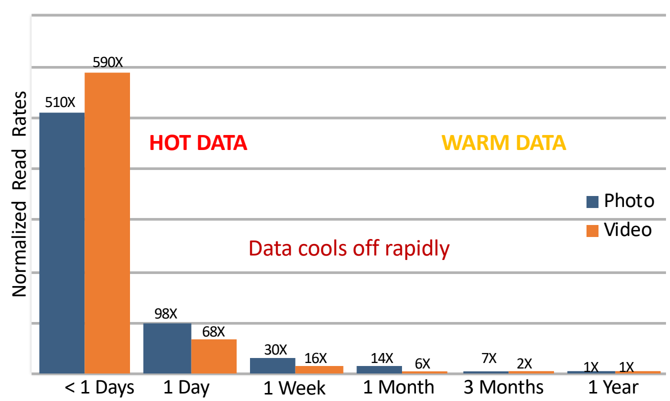
BLOBs@FB

Immutable
&
Unstructured

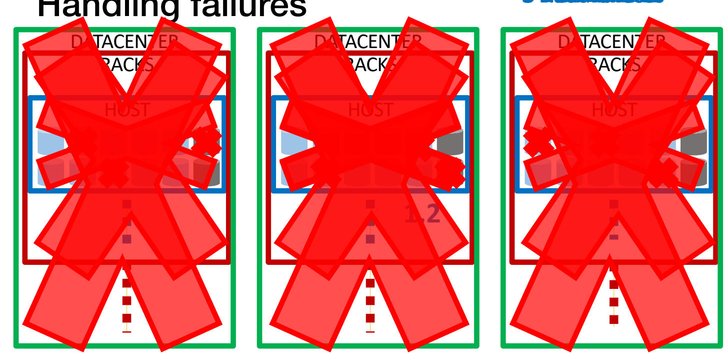
Diverse

A LOT of them!!



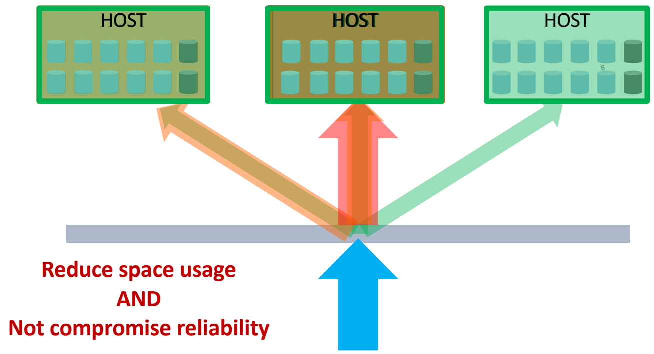


Handling failures



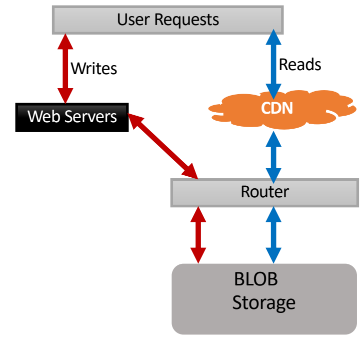
Replication: * 3 = 3.6

Handling load



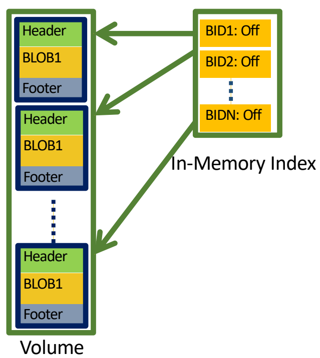
Background: Data serving

- CDN protects storage
- Router abstracts storage
- Web tier adds business logic

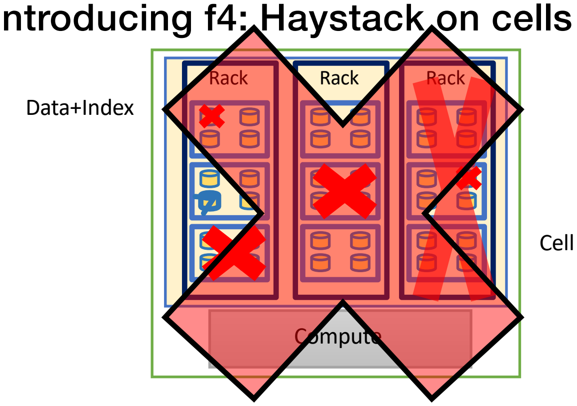


Background: Haystack [OSDI'10]

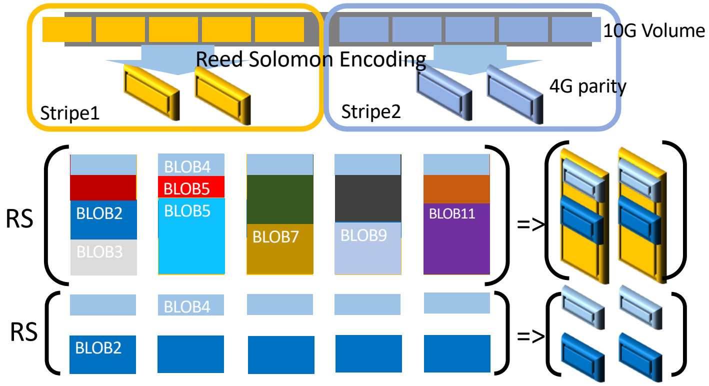
- Volume is a series of BLOBs
- In-memory index



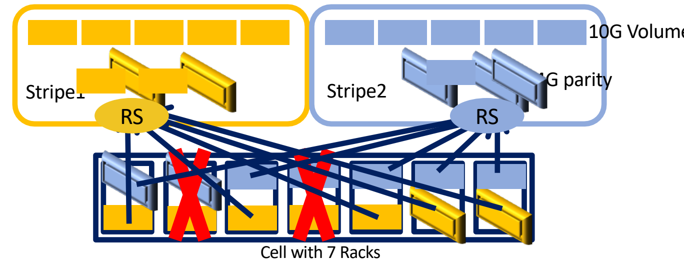
Introducing f4: Haystack on cells



Data splitting

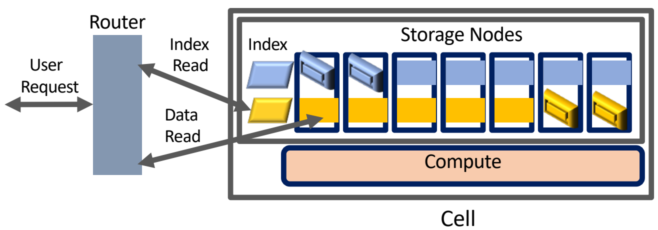


Data placement



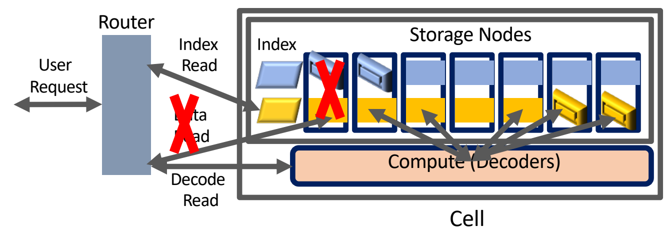
- Reed Solomon (10, 4) is used in practice (1.4X)
- Tolerates 4 racks (→ 4 disk/host) failures

Reads



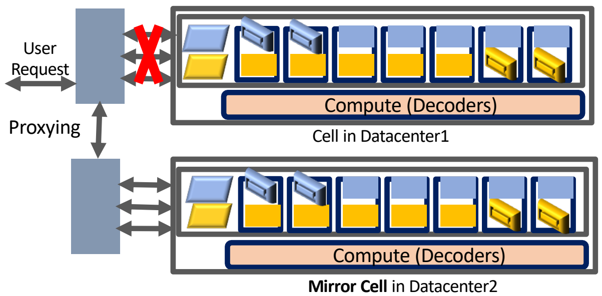
- 2-phase: Index read returns exact physical location of BLOB

Reads under cell-local failures



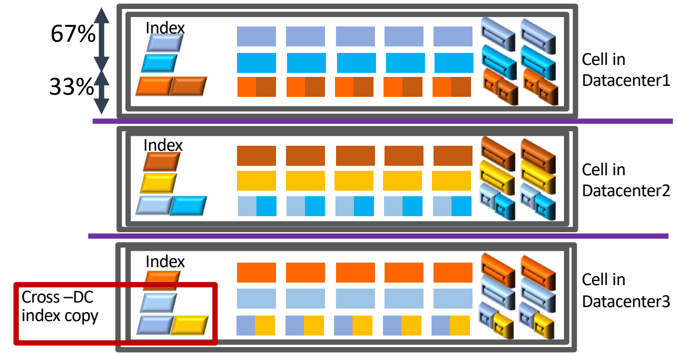
- Cell-Local failures (disks/hosts/racks) handled locally

Reads under datacenter failures (2.8X)

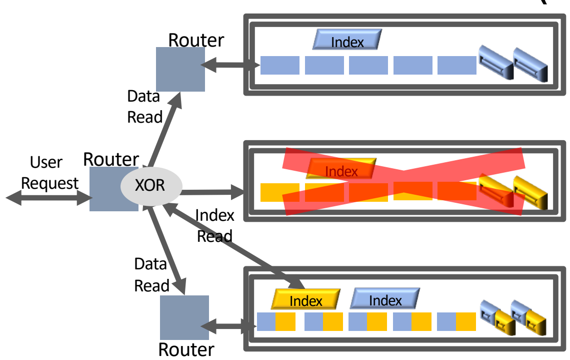


2 * 1.4X = 2.8X

Cross datacenter XOR (1.5 * 1.4 = 2.1X)



Reads with datacenter failures (2.1X)



	Haystack 3-way replication	f4 2.8 RS(10,4)	f4 2.1 RS(10,4)
Replication	3.6X	*2.8X*	*2.1X*
Irrecoverable Disk Failures	9	*10*	*10*
Irrecoverable Host Failures	3	*10*	*10*
Irrecoverable Rack failures	3	*10*	*10*
Irrecoverable Datacenter failures	*3*	2	2
Load split	*3X*	2X	1X

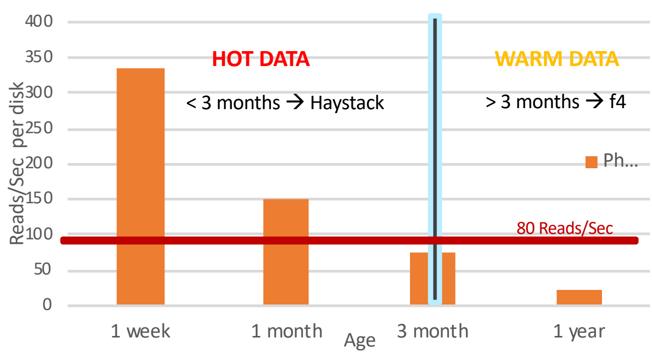
Evaluation

- What and how much data is "warm"?
- Can f4 satisfy throughput and latency requirements?
- How much space does f4 save
- f4 failure resilience

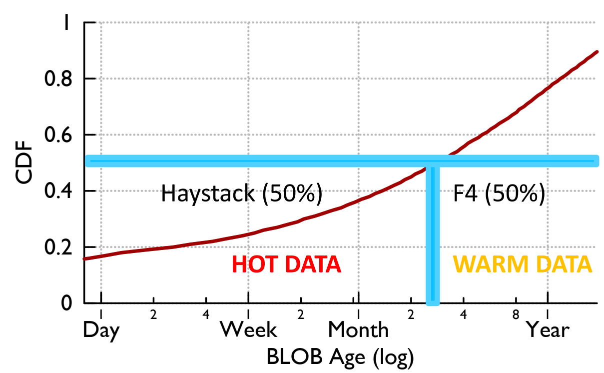
Methodology

- CDN data: 1 day, 0.5% sampling
- BLOB store data: 2 week, 0.1%
- Random distribution of BLOBs assumed
- The worst case rates reported

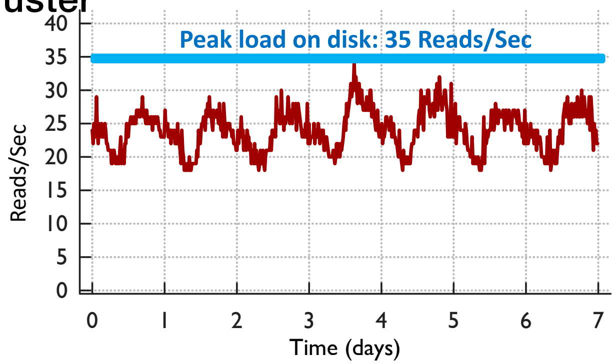
Hot and warm divide



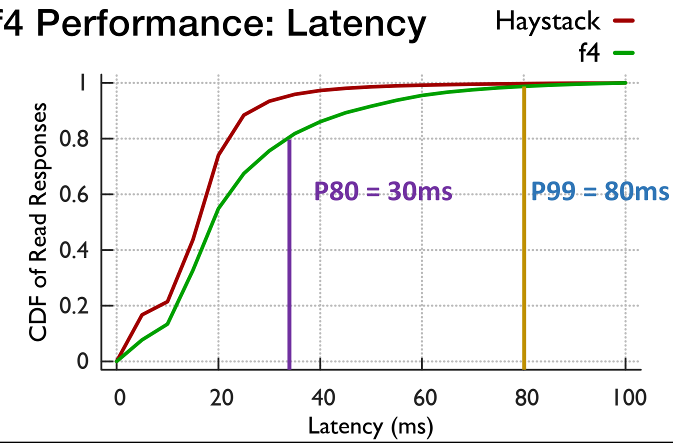
It is warm, not cold



f4 Performance: Most loaded disk in cluster



f4 Performance: Latency



Summary

- Facebook's BLOB storage is big and growing
- BLOBs cool down with age
    - ~100X drop in read requests in 60 days
- Haystack's 3.6X replication over provisioning for old, warm data.
- f4 encodes data to lower replication to 2.1X