SDC 18

September 24-27, 2018
Santa Clara, CA

www.storagedeveloper.org

QuadIron: an Open Source Library for Number Theoretic Transform-Based Erasure Codes

Vianney Rancurel
Giorgio Regni
Scality

Summary

- Introduction
- Properties of Codes
- Type of Codes
- Application
  - Decentralized Storage
- Using the Library

Introduction

What is QuadIron ?

- An open-source high performance erasure code library

Why ?

- Because we needed a large number of parities for world scale fault-tolerance

Why Open-source ?

- To benefit from a lot of pair of eyes
- Preparing to publish a paper

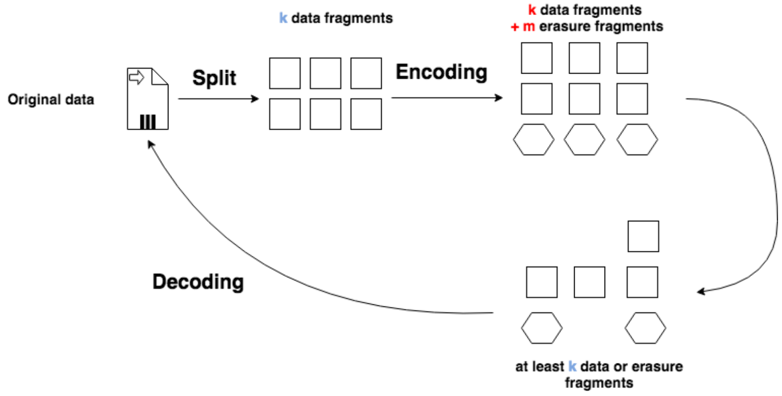
2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Properties of Erasure Codes: Definition

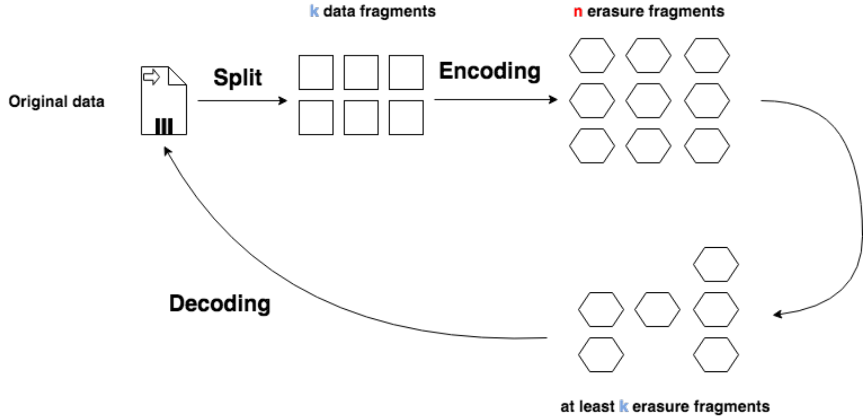
A C(n,k) erasure code is defined by n=k+m

- k being the number of data fragments.
- m being the number of desired erasure fragments.

Example: C(9, 6)



Systematic



Non-Systematic

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Properties of Erasure Codes

- **Optimality**: e.g. MDS (Maximum Distance Separable) erasure code guarantees that any k fragments can be used to decode a file
- **Systematicity**: Systematic codes generate n-k erasure fragments and therefore maintain k data fragments. Non-systematic codes generate n erasure fragments
- **Speed**: Erasure codes are characterized by their encode/decode speed. Speed may vary acc/to the rate (k and m parameters). Speeds may also be more or less predictive acc/to codes.
- **Rate sensitivity**: Erasure codes can also be compared by their sensitivity to the rate r=k/n, which may or may not impact the encoding and decoding speed
- **Rate adaptivity**: Changing k and m without having to generate all the erasure codes
- **Confidentiality**: determined if an attacker can partially decode the data if he obtains less than k fragments. Non-systematic codes are confidential (different from threshold schemes)
- **Repair Bandwidth**: the number of fragments required to repair a fragment.

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

(Main) Types of Erasure Codes

- Traditional RS Codes (e.g. Vandermonde or Cauchy matrices)
- LDPC Codes
- Locally-Repairable-Codes (LRC)
- FFT Based RS Codes
  - Multiplicative FFTs (prime fields)
  - Additive FFTs (binary extension fields)

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Types of Codes: Traditional RS Codes

A message $\mathbf{m} = (m_0, \ldots, m_k) \in \mathbb{F}_q^k$ can be represented as a polynomial $p_{\mathbf{m}}(x)$ of degree $k-1$:

$$p_{\mathbf{m}}(x) = m_0 + m_1 x + m_2 x^2 + \cdots + m_{k-1} x^{k-1} \tag{1}$$

The code word $\mathbf{c} = (c_0, \ldots, c_{n-1}) \in \mathbb{F}_q^n$ of the message $\mathbf{m}$ is obtained by evaluating $p_{\mathbf{m}}(x)$ at a given but arbitrary set of n different points $S = \{s_0, \ldots, s_{n-1}\}$ of the field $\mathbb{F}_q$, called the set of evaluation points. Concretely,

$$\begin{aligned} c_0 &= m_0 + m_1 s_0 + m_2 s_0^2 + \cdots + m_{k-1} s_0^{k-1}, \\ c_1 &= m_0 + m_1 s_1 + m_2 s_1^2 + \cdots + m_{k-1} s_1^{k-1}, \\ &\ldots \\ c_{n-1} &= m_0 + m_1 s_{n-1} + m_2 s_{n-1}^2 + \cdots + m_{k-1} s_{n-1}^{k-1} \end{aligned} \tag{2}$$

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Types of Codes: Traditional RS Codes

Equation (2) can be represented as a multiplication of the message $\mathbf{m}$ and matrix G:

$$\mathbf{c} = \mathbf{m} \times G \tag{3}$$

where G is the generator matrix of the RS codes:

$$G = \begin{bmatrix} 1 & 1 & \dots & 1 \\ s_0 & s_1 & \dots & s_{n-1} \\ s_0^2 & s_1^2 & \dots & s_{n-1}^2 \\ \vdots & \vdots & & \vdots \\ s_0^{k-1} & s_1^{k-1} & \dots & s_{n-1}^{k-1} \end{bmatrix} \tag{4}$$

Since $s_i \neq s_j$ when $i \neq j$, G is actually a Vandermonde matrix [8].

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Types of Codes: Traditional RS Codes

The good:

- Simple
- Support systematic and adaptive rates.

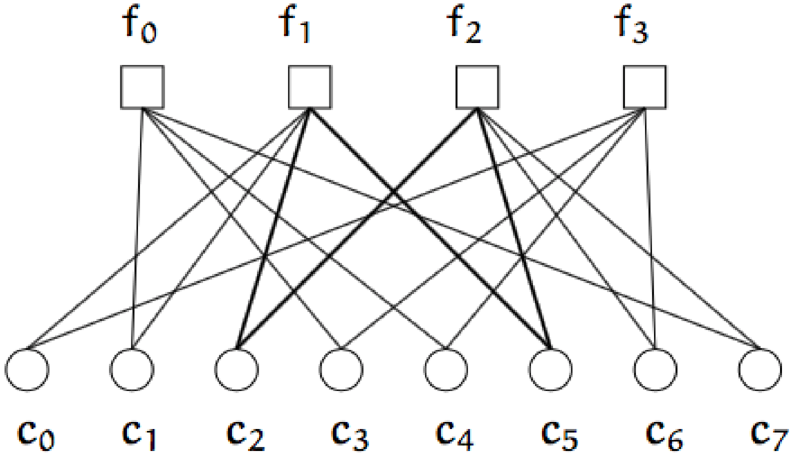
The bad:

- Matrix multiplication: O(k x n)

 © Scality Inc. All Rights Reserved.

Types of Codes: LDPC Codes

$$\mathbf{H} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$



- H is a matrix for a C(8,4) code
- w_c is the number of 1 in a col
- w_r is the number of 1s in a row
- To be called low density $w_c \ll n$ and $w_r \ll m$
- Regular if w_c constant and $w_r = w_c.(n/m)$
- Matrix can be generated pseudo-randomly
- Presence of short cycles f1, f2 bad

Source: Bernhard M.J. Leiner

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Types of Codes: LDPC Codes

Low-Density-Parity-Check (LDPC) codes are also an important class of erasure codes and are constructed over sparse parity-check matrices.

The good:

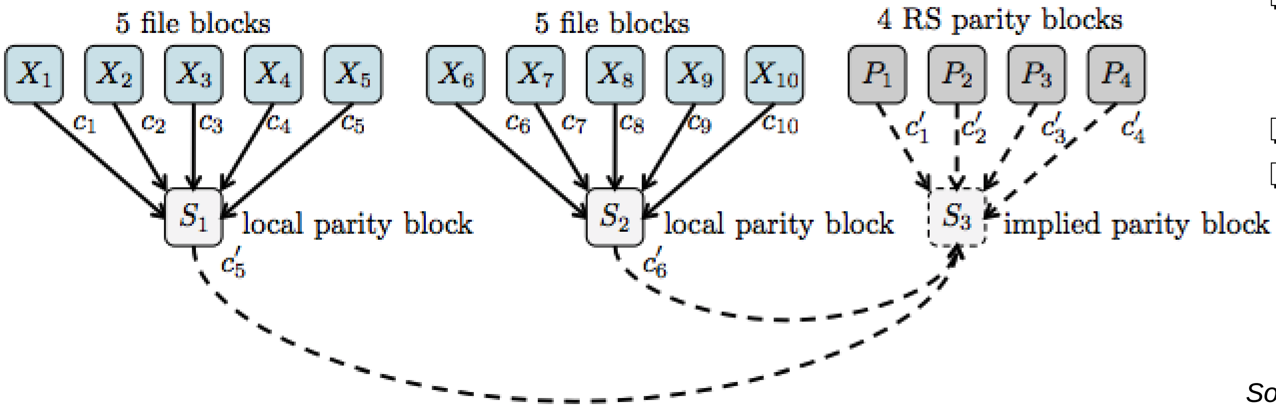
- Theoretically an LDPC code optimal for all the interesting properties for a given use case exist.

The bad:

- LDPC are not MDS: it is always possible to find a pattern that cannot decode (e.g. having only k fragments out of n). Overhead is k*f or k+f with a small f, but the overhead is not deterministic.
- You can always find/design an LDPC code optimized for few properties (i.e. tailored for a specific use case) but it will be sub-optimal for the other properties
- Designing a good LDPC code is some kind of black art that requires a lot of fine tuning and experimentation.

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Types of Codes: LRC Codes



- P1, P2, P3 and P4 are constructed over a standard RS
- S1 + S2 + S3 = 0
- No need to store S3

Source: XORing Elephants: Novel Erasure Codes for Big Data

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Types of Codes: LRC Codes

Locally-Repairable-Codes (LRC) have tackled the repair bandwidth issue of the RS codes. They combine multiple layers of RS: the local codes and the global codes.

The good:

- ❒ Better repair bandwidth than RS codes. Because with RS code we need to read k fragments to decode.

The bad:

- ❒ Those codes are not MDS and they require an higher storage overhead than MDS codes.

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Types of Codes: Multiplicative FFT

The set S is constructed as a multiplicative group whose generator is the N^{th} root of unity of the field $\mathbb{F}_q$, i.e.

$$S = \left\{\alpha^0, \ldots, \alpha^{N-1}\right\} \tag{11}$$

where α is the N^{th} root of unity of the field $\mathbb{F}_q$. We call this technique *multiplicative FFT*.

The FFT technique applied for the constructed set S was first introduced in [9]. Supposing that $N = N_1 \times N_2$, the FFT operation on $\mathbf{m}$ is split into two FFT operations on two vectors $\mathbf{m}_1$ and $\mathbf{m}_2$ of length N_1 and N_2 respectively. This results in a computational complexity $\mathcal{O}(N(N_1 + N_2))$. If N is highly composite, this technique reduces the computation time to $\mathcal{O}(N \log N)$.

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Types of Codes: Multiplicative FFT

To obtain such advantages, however, this technique requires the satisfaction of two conditions:

1) N is a divisor of $q - 1$ (for the existing N^{th} root of unity of the field $\mathbb{F}_q$),
2) N is highly composite, e.g. $N = 2^v$ ideally

A simple solution to satisfying these conditions is to use the field $\mathbb{F}_{q=p^w}$ where p is prime and $p - 1$ is highly composite. Fermat numbers, i.e. $F_i = 2^{2^i} + 1$ for $i \leq 4$, are perfectly suitable for that. This technique, also called Fermat Number Transform (FNT) based erasure codes, was introduced and analyzed in [10], [11].

SDC 18

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Types of Codes: Additive FFT

This technique, however, does not efficiently apply to the binary extension field $\mathbb{F}_{2^w}$ because divisors of $(2^w - 1)$ are not highly composite. However, $\mathbb{F}_{2^w}$ is perfectly suitable for most practical applications because each element can be expressed by n bits. The following FFT technique focuses on this field.

The FFT length is necessarily a power of 2, i.e. $N = 2^m$ with $m \leq w$. Let $\beta_0, \ldots, \beta_{m-1}$ be m linearly independent elements of $\mathbb{F}_{2^n}$. The set S is chosen as a subspace spanned by β_i over $\mathbb{F}_2$, i.e.

$$s_i = i_0\beta_0 + i_1\beta_1 + i_2\beta_2 + \cdots + i_{m-1}\beta_{m-1}, \quad \text{for } 0 \leq i \leq 2^m - 1 \qquad (12)$$

where $i = i_0 + i_1 2 + i_2 2^2 + \cdots + i_{m-1}2^{m-1}$ with $i_j \in \mathbb{F}_2$. The FFT operation on a vector $\mathbf{m}$ is re-expressed as two FFTs on two vectors of half length.

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Types of Codes: FFT Based RS Codes

Fast Fourier transform (FFT) have a good set of desirable properties.

The good:

- Relatively simple
- O(N.log(N)) (because we use FFT to speed up the matrix multiplication)
- MDS
- Fast for large n

The bad:

- Repair bandwidth: If there is a missing erasure, we need k codes to recover the data fragments. For systematic codes, in any case we need to download k codes.

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Multiplicative FFT: Vectorization

$$\begin{aligned}\vec{a} &= (a_1, a_2, \dots, a_n) \\ \vec{b} &= (b_1, b_2, \dots, b_n) \\ \vec{c} &= (c_1, c_2, \dots, c_n)\end{aligned} \tag{13}$$

- Addition

$$c_i = (a_i + b_i)\%q, \text{ for } i = 1, \dots, n$$

- Subtraction

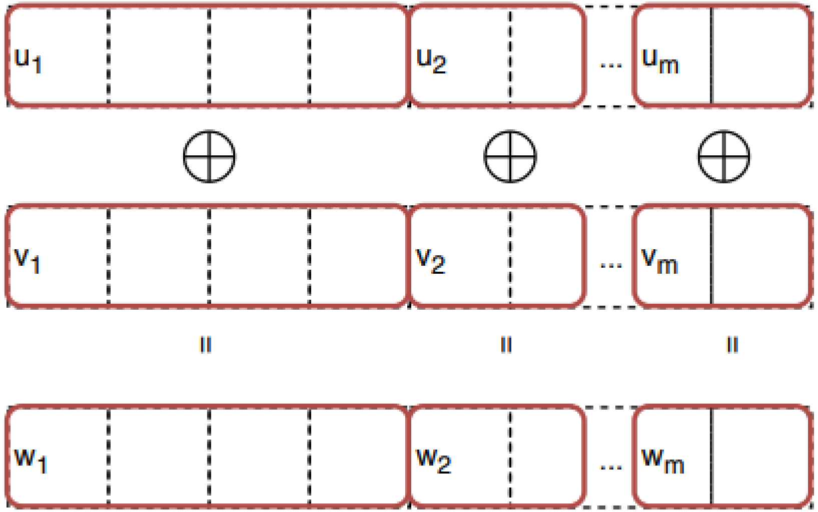
$$c_i = (a_i - b_i)\%q, \text{ for } i = 1, \dots, n$$

- Hadamard multiplication

$$c_i = (a_i * b_i)\%q, \text{ for } i = 1, \dots, n$$

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Multiplicative FFT: Horizontal Vectorization



2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Multiplicative FFT: Vertical Vectorization

$a_{1,1}$	$a_{2,1}$	$a_{3,1}$	$a_{4,1}$	$a_{5,1}$	...	$a_{n,1}$
$a_{1,2}$	$a_{2,2}$	$a_{3,2}$	$a_{4,2}$	$a_{5,2}$	...	$a_{n,2}$
$a_{1,3}$	$a_{2,3}$	$a_{3,3}$	$a_{4,3}$	$a_{5,3}$	...	$a_{n,3}$
$a_{1,4}$	$a_{2,4}$	$a_{3,4}$	$a_{4,4}$	$a_{5,4}$	...	$a_{n,4}$
$a_{1,5}$	$a_{2,5}$	$a_{3,5}$	$a_{4,5}$	$a_{5,5}$	...	$a_{n,5}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$a_{1,s}$	$a_{2,s}$	$a_{3,s}$	$a_{4,s}$	$a_{5,s}$	...	$a_{n,s}$

$\oplus$

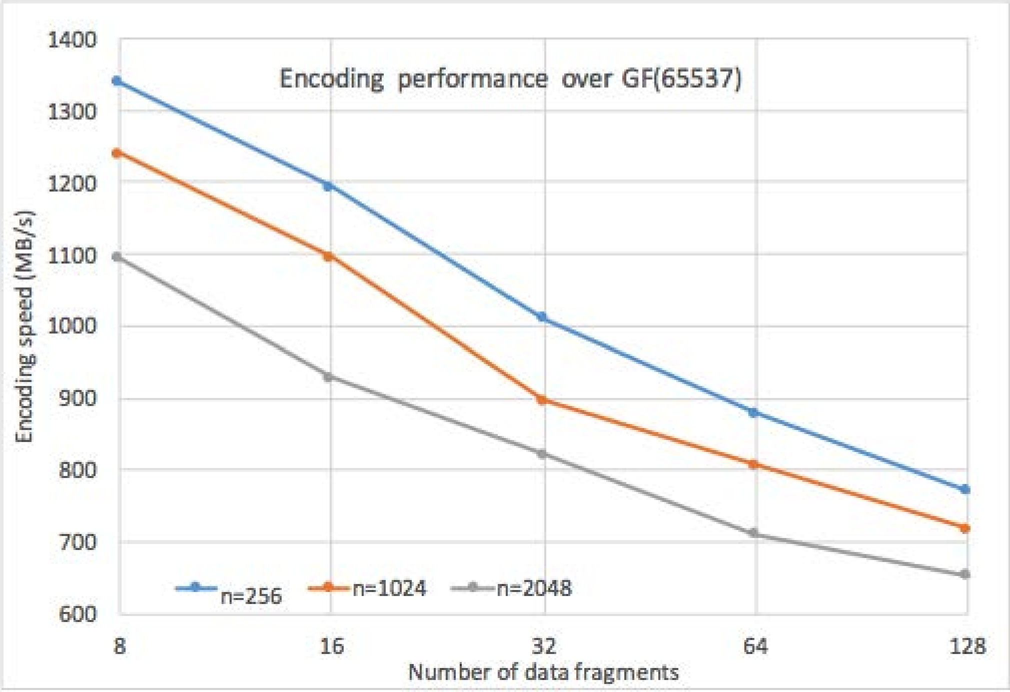
$b_{1,1}$	$b_{2,1}$	$b_{3,1}$	$b_{4,1}$	$b_{5,1}$	...	$b_{n,1}$
$b_{1,2}$	$b_{2,2}$	$b_{3,2}$	$b_{4,2}$	$b_{5,2}$	...	$b_{n,2}$
$b_{1,3}$	$b_{2,3}$	$b_{3,3}$	$b_{4,3}$	$b_{5,3}$	...	$b_{n,3}$
$b_{1,4}$	$b_{2,4}$	$b_{3,4}$	$b_{4,4}$	$b_{5,4}$	...	$b_{n,4}$
$b_{1,5}$	$b_{2,5}$	$b_{3,5}$	$b_{4,5}$	$b_{5,5}$	...	$b_{n,5}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$b_{1,s}$	$b_{2,s}$	$b_{3,s}$	$b_{4,s}$	$b_{5,s}$	...	$b_{n,s}$

=

$c_{1,1}$	$c_{2,1}$	$c_{3,1}$	$c_{4,1}$	$c_{5,1}$	...	$c_{n,1}$
$c_{1,2}$	$c_{2,2}$	$c_{3,2}$	$c_{4,2}$	$c_{5,2}$	...	$c_{n,2}$
$c_{1,3}$	$c_{2,3}$	$c_{3,3}$	$c_{4,3}$	$c_{5,3}$	...	$c_{n,3}$
$c_{1,4}$	$c_{2,4}$	$c_{3,4}$	$c_{4,4}$	$c_{5,4}$	...	$c_{n,4}$
$c_{1,5}$	$c_{2,5}$	$c_{3,5}$	$c_{4,5}$	$c_{5,5}$	...	$c_{n,5}$
⋮	⋮	⋮	⋮	⋮	⋮	⋮
$c_{1,s}$	$c_{2,s}$	$c_{3,s}$	$c_{4,s}$	$c_{5,s}$	...	$c_{n,s}$

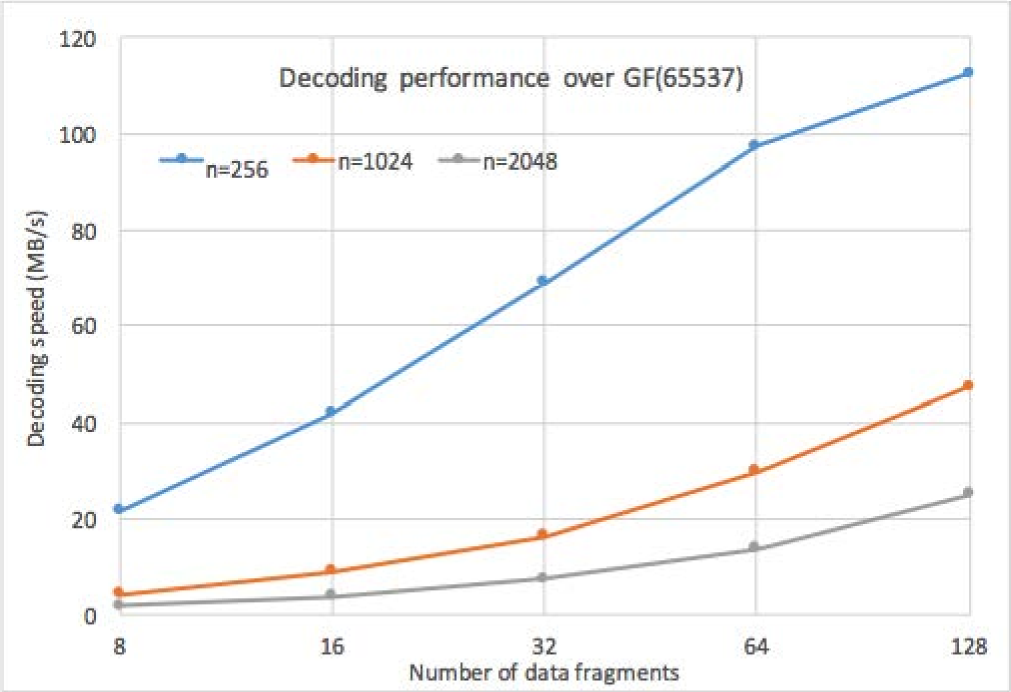
2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Multiplicative FFT: Vertical Vectorization



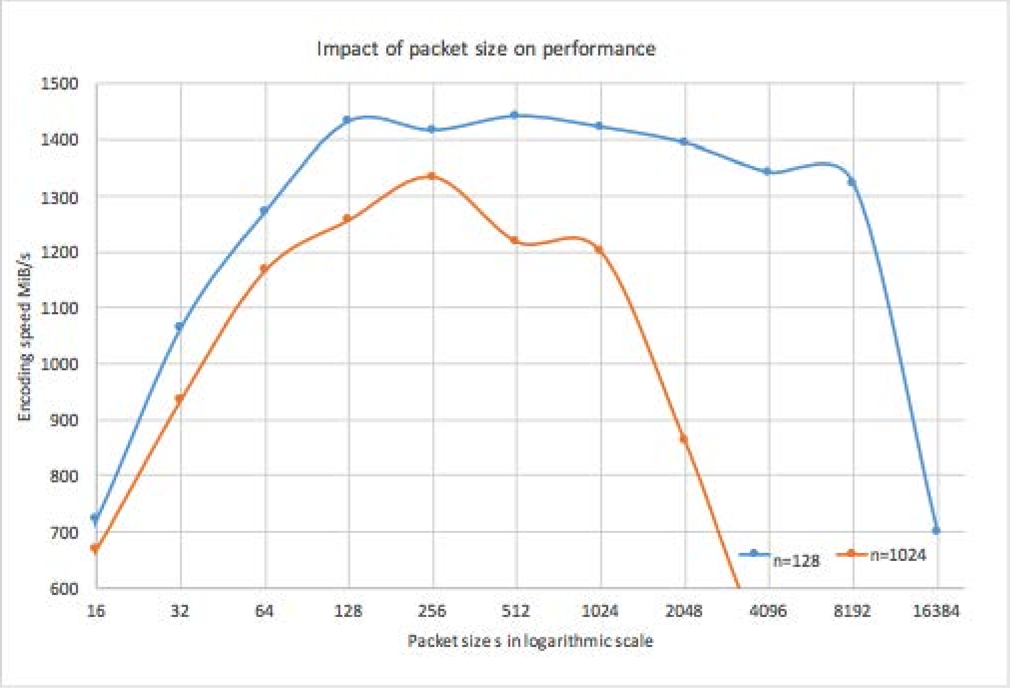
2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Multiplicative FFT: Vertical Vectorization



2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Multiplicative FFT: Vertical Vectorization



2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

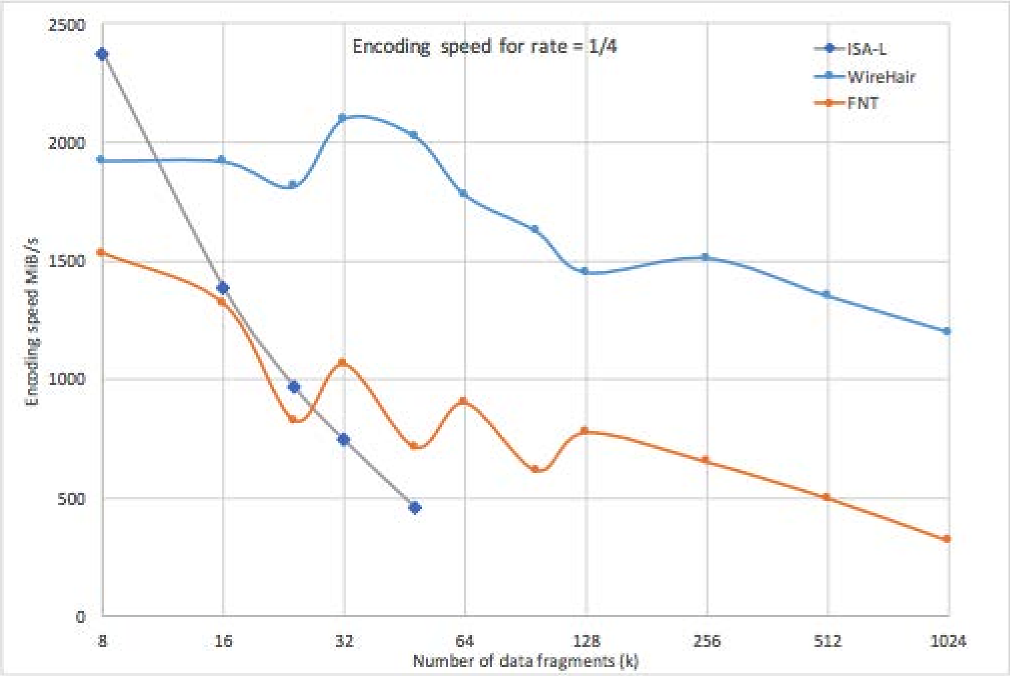
Speed Comparison

- Isa-l: Intel Intelligent Storage Acceleration Library. Matrix based RS HW accelerated: http://01.org/intel-storage-acceleration-library-open-source-version
- Wirehair: Fast and Portable Fountain Codes in C. Hybrid LDPC. https://github.com/catid/wirehair
- Leopard: MDS Reed-Solomon Erasure Correction Codes for Large Data in C. Additive FFT based. https://github.com/catid/leopard

Thanks Catid !

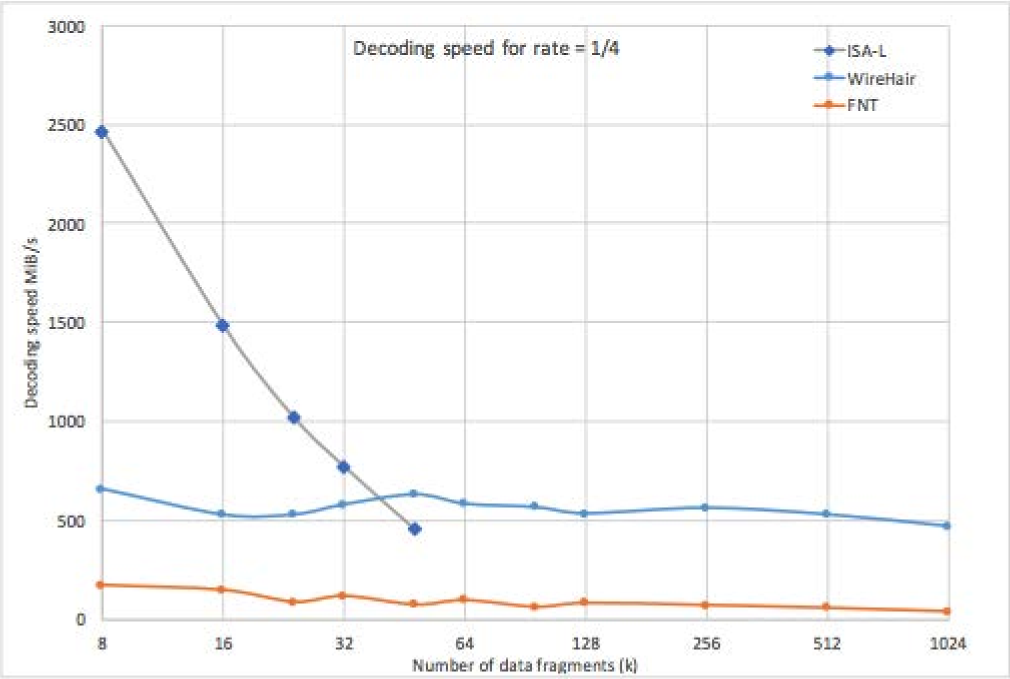
2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Types of Codes: Speed Comparison



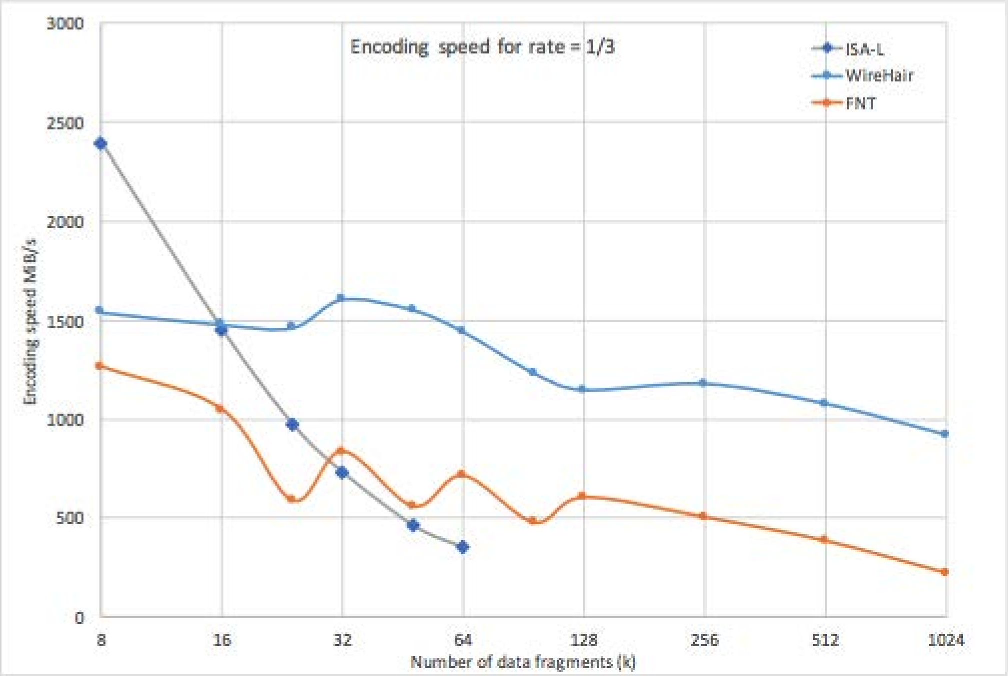
2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Types of Codes: Speed Comparison



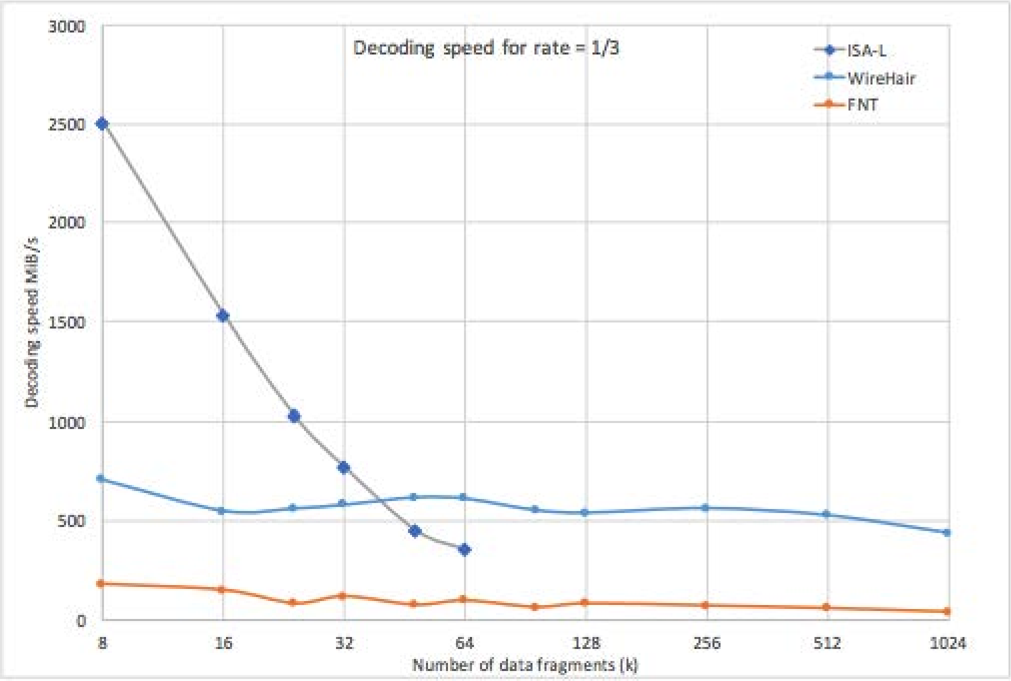
2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Types of Codes: Speed Comparison



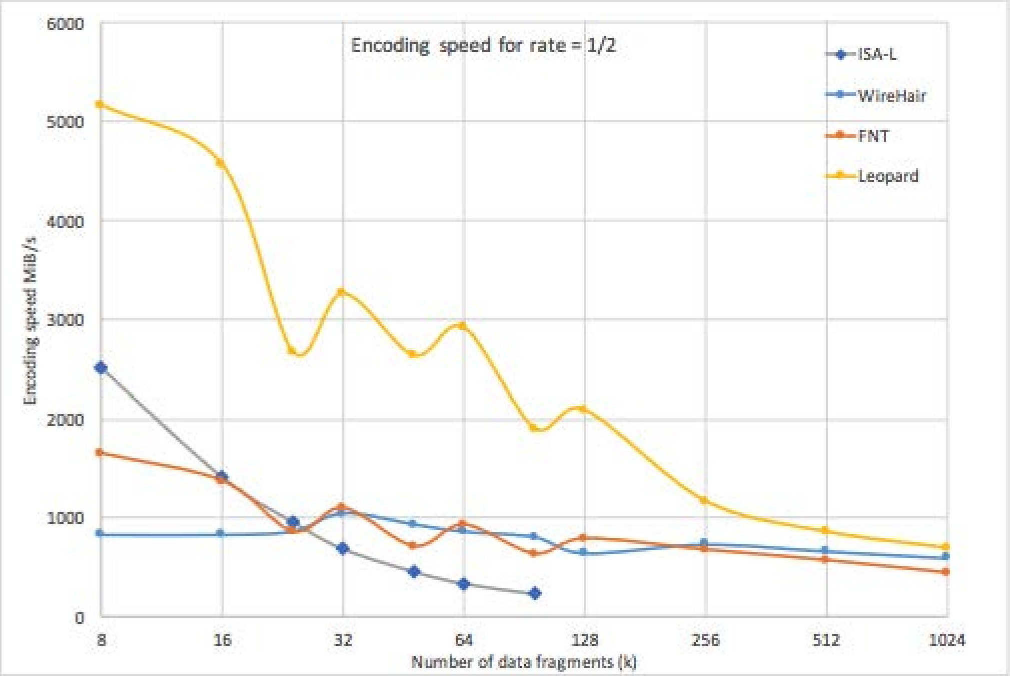
2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Types of Codes: Speed Comparison



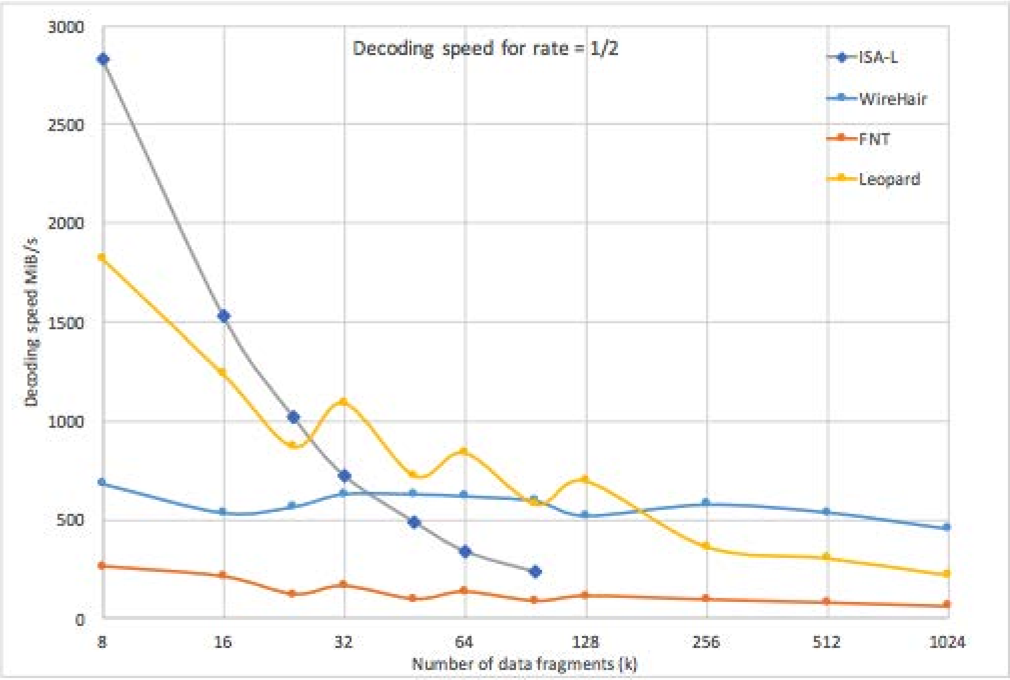
2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Types of Codes: Speed Comparison



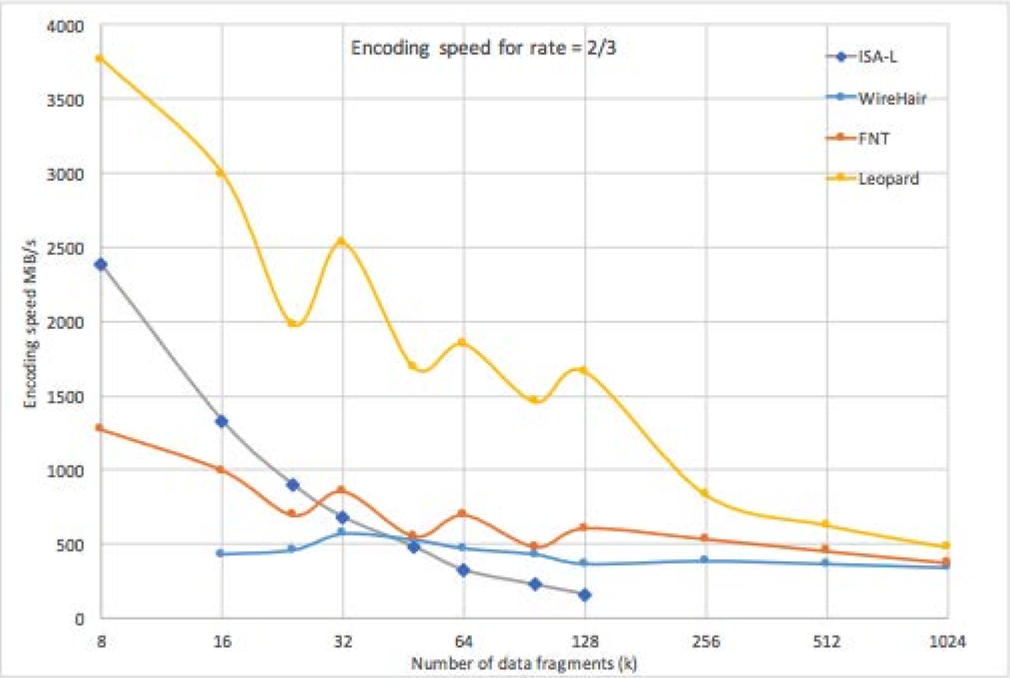
2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Types of Codes: Speed Comparison



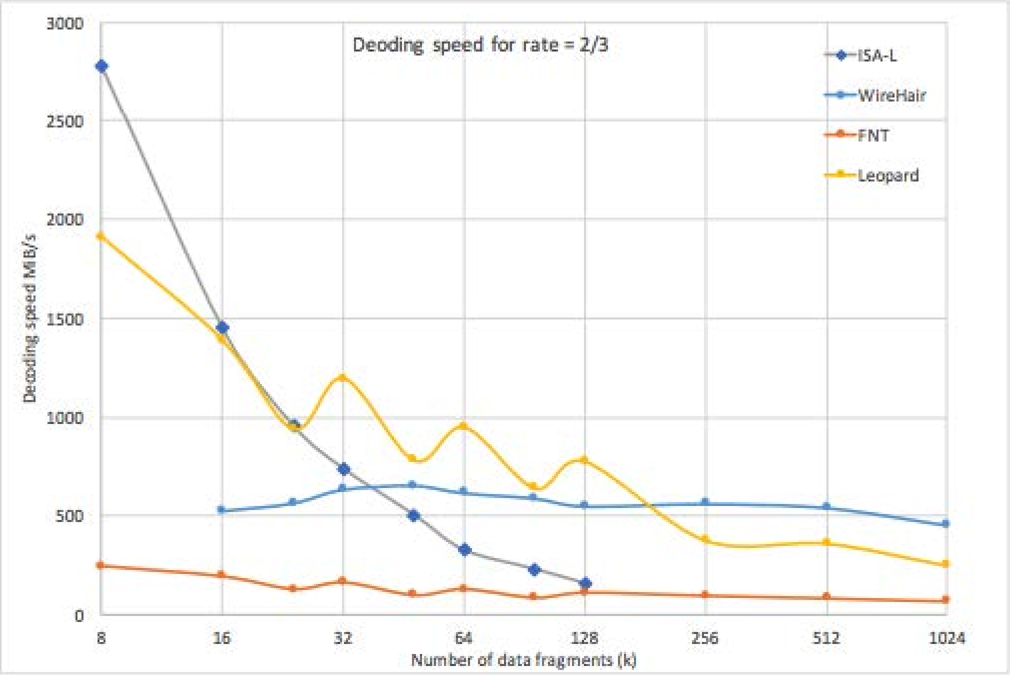
2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Types of Codes: Speed Comparison



2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Types of Codes: Speed Comparison



2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Application

- Decentralized Storage

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

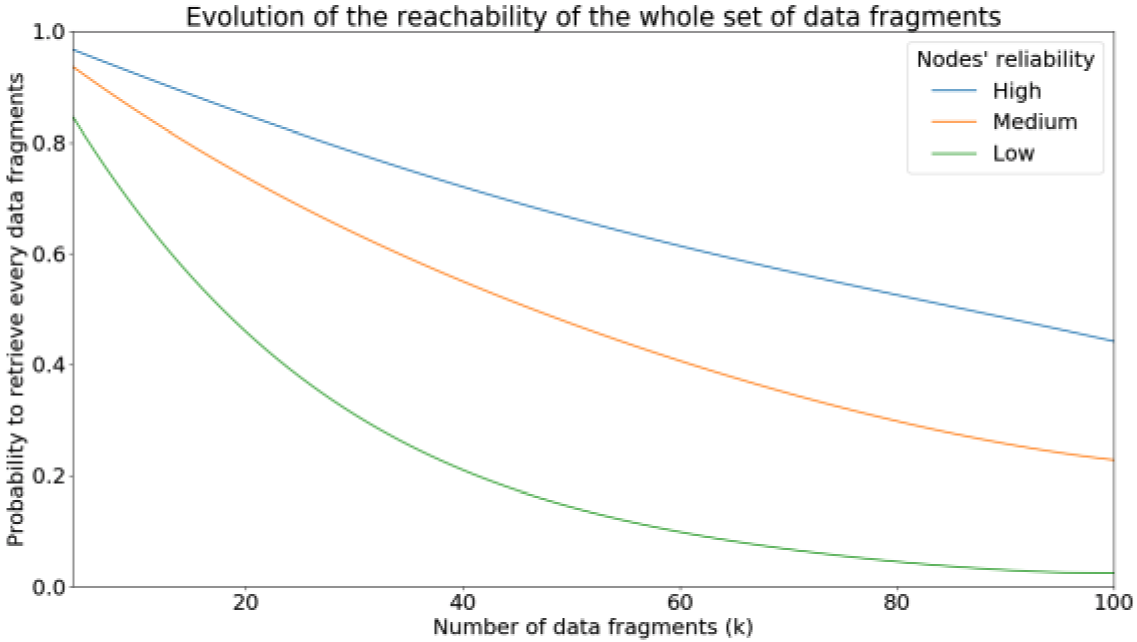
Application: Decentralized Storage

Requirements for an erasure code for a decentralized storage archive:

- ❒ Simple (e.g. may compile on WASM)
- ❒ Fast, e.g. for > 24 fragments
- ❒ MDS: A rock solid contract
- ❒ Work with all rates, and all combinations of n and k
- ❒ Systematic for smaller fragments
- ❒ Non-systematic for larger fragments -> Confidentiality ensured if fragments not stored on same servers (not a threshold scheme though, must be combined with encryption)
- ❒ Repair-Bandwidth not critical

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

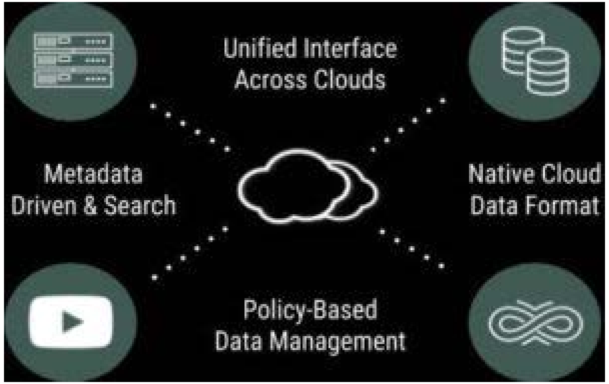
Application: Decentralized Storage

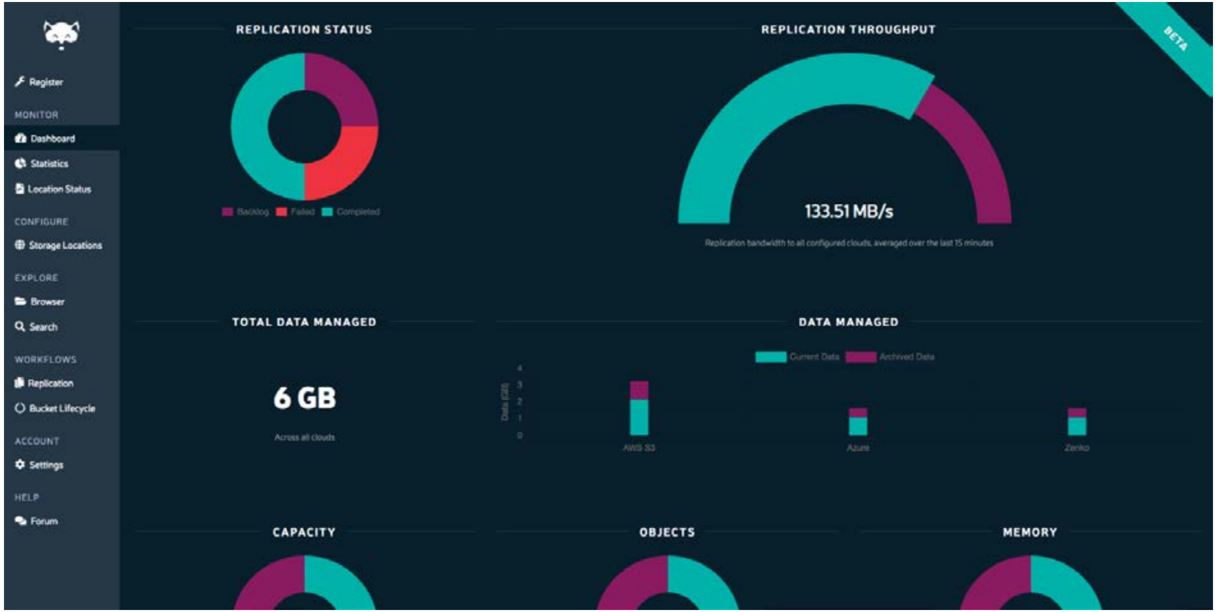


2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Applications: Use Case

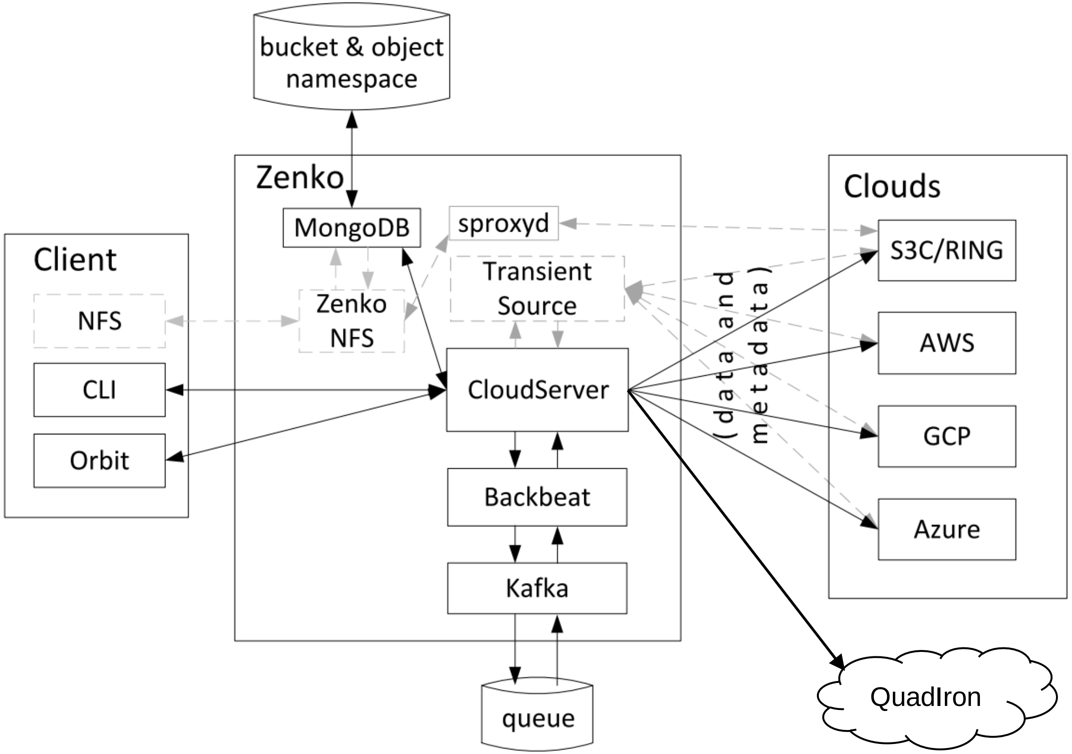
ZENKO





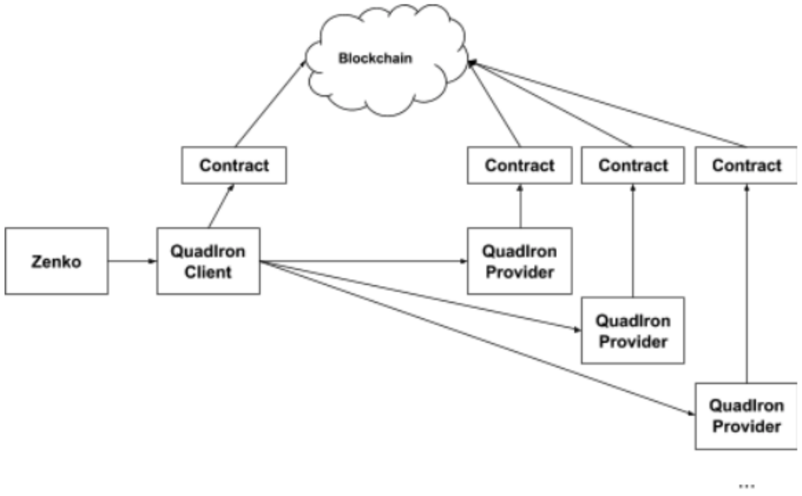
2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Decentralized Storage: Zenko



2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Application: Decentralized Storage



- Multiple locations, multiple servers per location
  - Each server is a “Quadiron Provider”
  - E.g. 10 locations on the globe with 5 servers/location: C(50,35) => can lose 3 locations or 15 servers for an overhead of 1.4
  - A server is just a bunch of disks, e.g. 45 drives
  - Can have local parities on servers to avoid repairing too often on the network e.g. C(45, 40) = 1.125
  - Total overhead 1.4 * 1.125 = 1.57
  - E.g. w/ 10TB drives, 22PB => 14PB useful
- Use blockchain transactions to store the location of blocks
  - E.g. using Parity, proof-of-work (non-trusted env) or proof-of-authority (trusted env => millions tx/s)
  - Index the ledges by block-ids
  - Use the indexes to locate the blocks
  - Consolidate indexes

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Using the Library

C++ Library is available at: https://github.com/scality/quadiron
LICENSE: BSD 3-clause

Compiling:

```
$ mkdir build
$ cd build
$ cmake -G 'Unix Makefiles' ..
$ make
```


2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Using the Library: Code

```
// #include <quadiron.h>

const int word_size   = 8;
const int n_data      = 16;
const int n_parities  = 64;
const size_t pkt_size = 1024;

quadiron::fec::RsFnt<T>* fec = new quadiron::fec::RsFnt<uint64_t>(
    quadiron::fec::FecType::NON_SYSTEMATIC, word_size, n_data, n_parities, pkt_size
);

// encode
std::vector<std::istream*>        d_files(fec->n_data, nullptr);
std::vector<std::ostream*>        c_files(fec->n_outputs, nullptr);
std::vector<quadiron::Properties> c_props(fec->n_outputs);
fec->encode_packet(d_files, c_files, c_props);

// decode
std::vector<std::ostream*> r_files(fec->n_data, nullptr);
fec->decode_bufs(d_files, c_files, c_props, r_files);
```

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Using the Library: Next Steps

- Optimize Multiplicative FFT decoding: For now a relatively slow Lagrange interpolation
  - We know how to do it for special values of k and m (k mod m = 0, m mod k = 0)
- Optimize Additive FFTs
- Implement Systematic Additives FFTs
- Implement NTT adaptive codes for both multiplicative and additive FFTs
- Other optimizations
  - Frobenius FFTs for both multiplicative and additive FFTs

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Developers

Lam Pham-Sy: Lam Pham-Sy is a research engineer working on information theory and computer science. His main research focuses on different families of forward erasure correcting codes such as ReedSolomon codes, Low-Density Parity-Check codes, Locally Repairable codes etc. Their application covers from digital communication to data storage. He did his PhD program in a collaboration between CEA-Leti and Eutelsat S.A. on the subject of forward erasure codes for satellite communications. Afterwards, he continued his researches at ETIS laboratory and at Orange Labs. Currently he works at Scality S.A. as a research engineer whose research topics include application of erasure codes in distributed storage systems, finite field arithmetics.

Sylvain Laperche: Sylvain Laperche is a code craftsman. With a background in biotech engineering,he learnt how to hack bacteria before learning how to hack a computer. That changed when it studied bioinformatics, and since then he honed and applied its skill on a wide set of problematics: genome sequencing, complex embedded systems, climate modelling at European scale, mass-scale geolocation for telco industries. Its steps led him to work on distributed storage systems and he currently works as an R&D engineer at Scality. Sylvain Laperche has an Engineer’s degree in Hardware, Circuit Design and Embedded Systems from ISIMA.

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Zenko/QuadIron Community

1,000+ registered Zenko Orbit users

Forum https://forum.zenko.io
Website: www.zenko.io/blog
QuadIron github: https://github.com/scality/quadiron

https://www.zenko.io/blog/free-library-erasure-codes/

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.

Questions ?

2018 Storage Developer Conference. © Scality Inc. All Rights Reserved.