Efficient Reliability in Volunteer Storage Systems with Random Linear Coding

Adam Visegradi, Peter Kacsuk

Resilience workshop, Euro-Par 2014, Porto

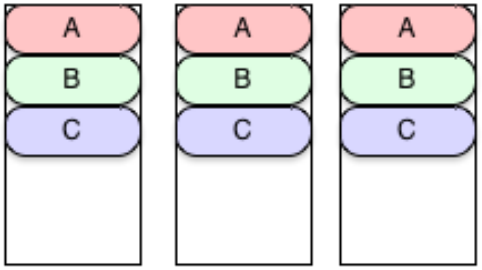
Motivation: Desktop Grids

- Volunteer, thus cheap and unreliable resources
  - Trivially parallelizable computation
- Node failure is expected
  - Majority of resources being inaccessible is b.a.u.
  - CPU power is cheap, replication is a good solution
- Not suitable for data storage, data intensive applications—yet
  - Storage limitations
    - Little storage on nodes—small capacity
    - Cannot *buy* storage; available capacity is determined by volunteers
  - High failure rate

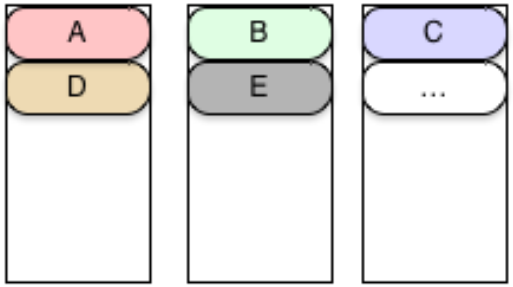
Reliability in Distributed Storage Systems

- Storing data on distributed nodes, nodes may fail
  - Need *redundancy* to cope with node failures
- Storage is costly or otherwise limited
  - Need *efficient* redundancy—best redundancy/reliability ratio
- Generic problems, not DG specific

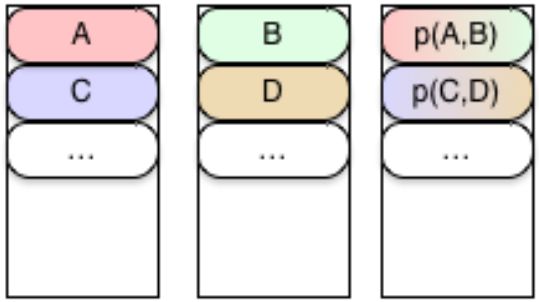
Data Storage



Replication



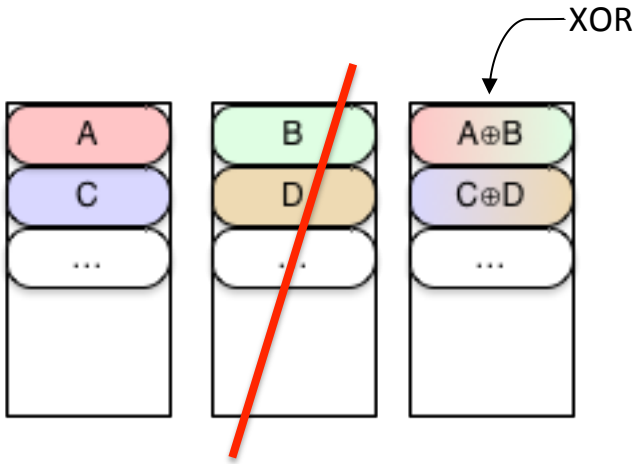
Striping



Striping and parity

Example

Striping+Parity



$$\underline{A} \oplus (\underline{A \oplus B}) = (A \oplus A) \oplus B = 0 \oplus B = \underline{B}$$

Erasure Coding

- Generally (n, k)
  - k original blocks: $b_1, b_2, \ldots, b_k$
  - n-k parity blocks: $p_1, p_2, \ldots, p_{(n-k)}$
  - In total, n blocks: $b_1, \ldots, b_k, p_1, \ldots, p_{(n-k)}$; **all different**
  - In such a way that *any* k blocks can be used to reconstruct the original data

- This schema can withstand the erasure of any (n-k) blocks

Erasure Coding

- Much more efficient than replication
  - In terms of redundancy/reliability ratio
    - Reason: in replication, P[all copies of a block is lost] > 0
    - E.g.: 40 blocks, need to withstand 20 failures
      - Erasure coding: 50%; replication: 2000% [1]
  - k|n is not required (unlike replication)

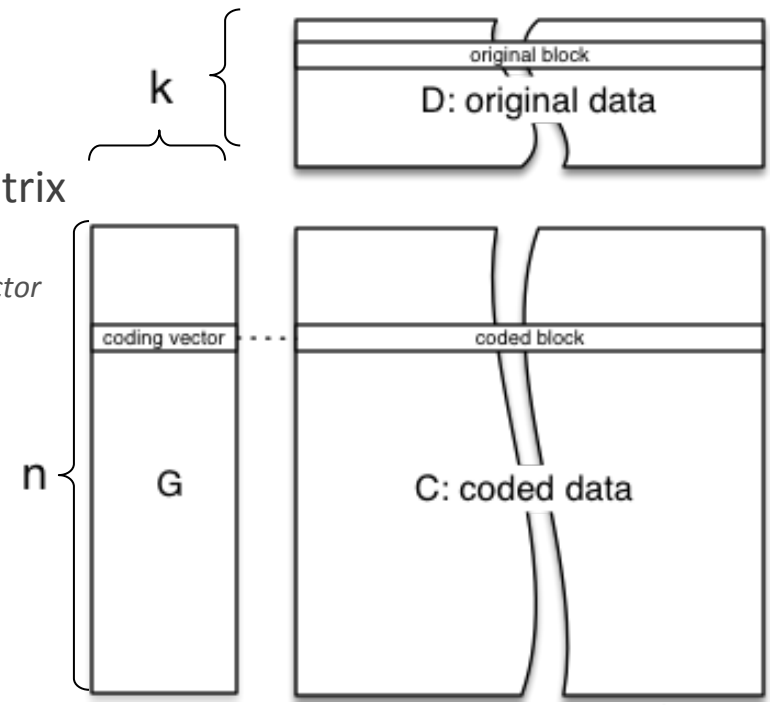
- Drawbacks
  - (De)coding requires memory
    - Data must be chunked in sufficiently small segments (several MB)
  - CPU-intensive
    - Preferable for cold data or data processing in steady streams
    - Not suitable for random access applications

[1] http://boinc.berkeley.edu/trac/wiki/VolunteerDataArchival

Linear Erasure Coding

- (n,k) **linear** erasure coding of *k* data blocks

- Based on a pre-determined *generator matrix* over a finite field
  - $G \in GF(q)^{n \times k}$ — *n* rows, *k* columns
- Data is considered to be a matrix of *k* blocks
  - $D \in GF(q)^{k \times b}$ — *b* is irrelevant
- **Coding** is multiplying the data with the generator matrix
  - $C := G \cdot D$
  - Resulting in *n* coded blocks, each pertaining to a row of G—its *coding vector*
  - $Gx=C$ is an overdetermined L.E.S. with unique solution
- Erasing **any** *n-k* equations from this system
  - (G', C') := remaining equations' matrices
  - G' is a square matrix ($k \times k$)
- **Decoding** is solving a L.E.S.
  - $G'x = C'$ is a solvable L.E.S., where $x = D$



Linear Erasure Coding

- Coding and decoding requires a generator matrix, G
  - It has to be determined **in advance**
    - Gx=C must be uniquely solvable
    - Any size-*k* subset of G'x=C' must be solvable
  - Usually Cauchy or Vandermonde matrix
- G, and thus, *n* are fixed
  - Rateful coding
  - Not adaptable
- Reparation of the system requires decoding the original data

Random Linear Coding

- The elements of G are randomly selected, thus
  - G is not determined a-priori
  - G is resizable: rateless coding
  - —and feasible with high probability
    - Any subset G'x=C' is solvable *with high probability*

Why use RLC?

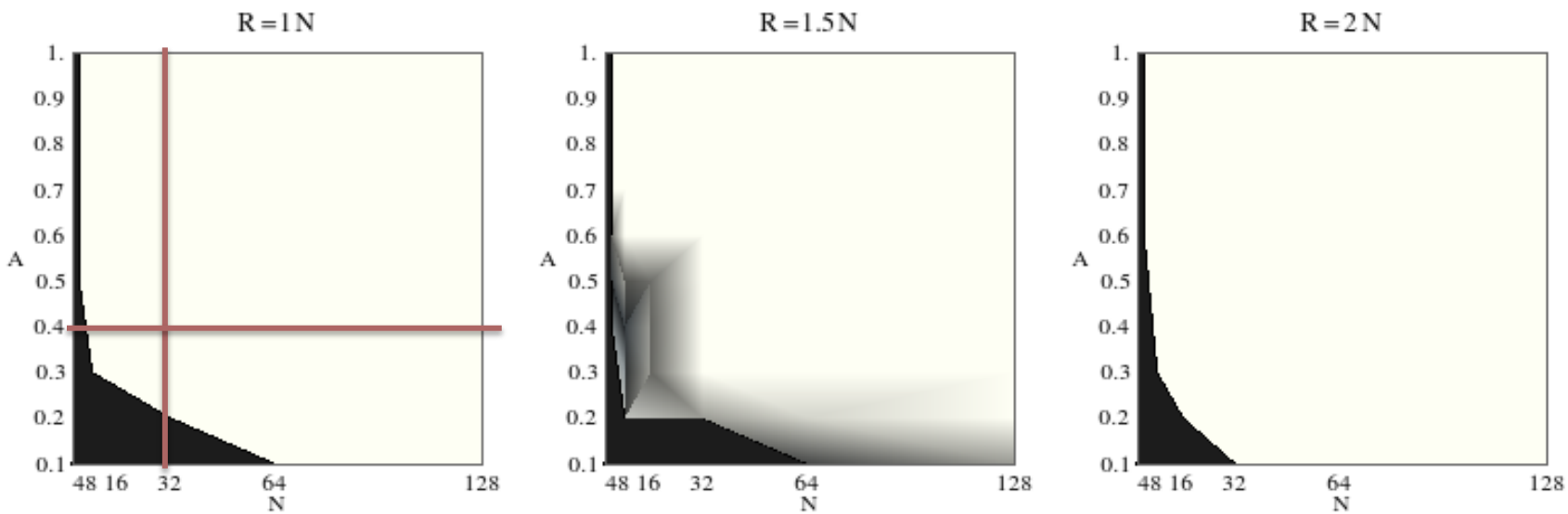
- Inherits everything from EC
  - Low redundancy/reliability ratio
  - CPU-intensive
- Randomization
  - Adaptable
    - Rateless coding ➔ redundancy can be scaled on-demand
  - Negligible control overhead
  - Iterative reparation
    - New coded block := linear combination of old coded blocks
    - No decoding is needed for reparation
  - Sparse coding
    - Only $A \cdot k$ elements of a coding vector are non-zero; A: sparsity
    - Less CPU-intensive coding than non-sparse EC
    - Also reduces transfer overhead for reparation

Measurements

- Metric
  - Reliability
  - Transfer overhead caused by stochastic property
- Focus on
  - Sparse coding—addressing CPU intensiveness
  - Iterative coding—addressing reparation problem
  - *Highly* unreliable nodes, as per desktop grids

Results

Reliability as a function of sparsity

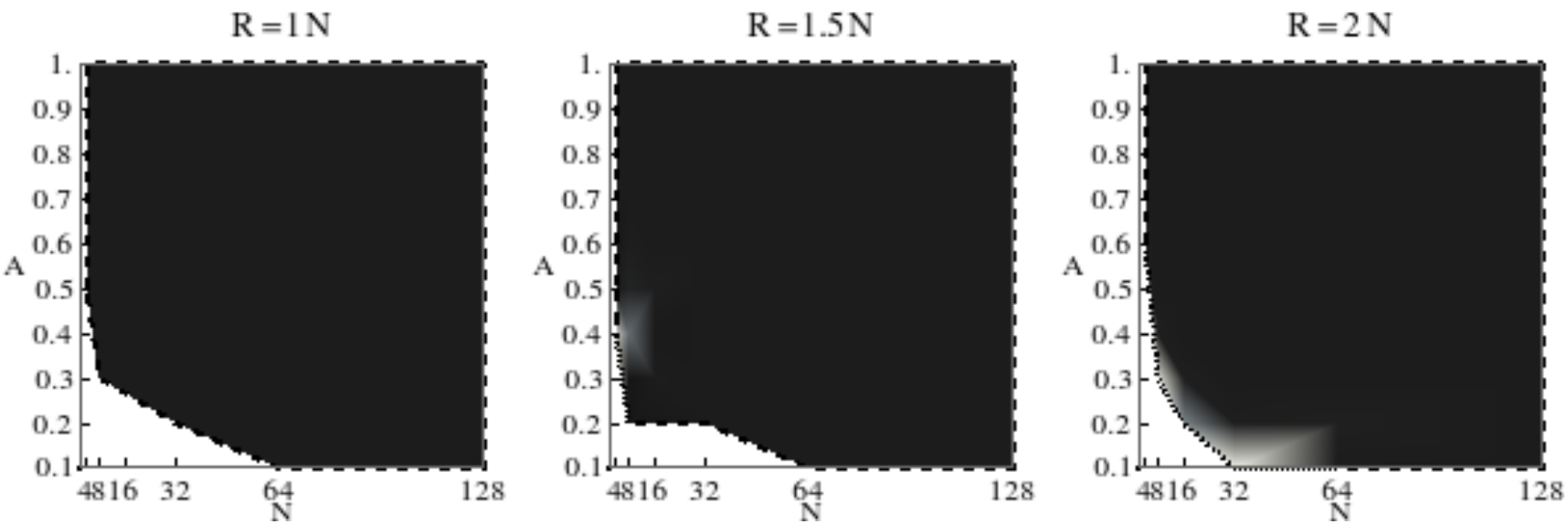


N: number of blocks of a file
A: sparsity factor; ratio of fixed 0 elements (A=1: no sparse coding is used)
R: redundancy; total number of stored blocks

$GF(q) = GF(2^{16})$

Results

Wasted bandwidth as a function of sparsity

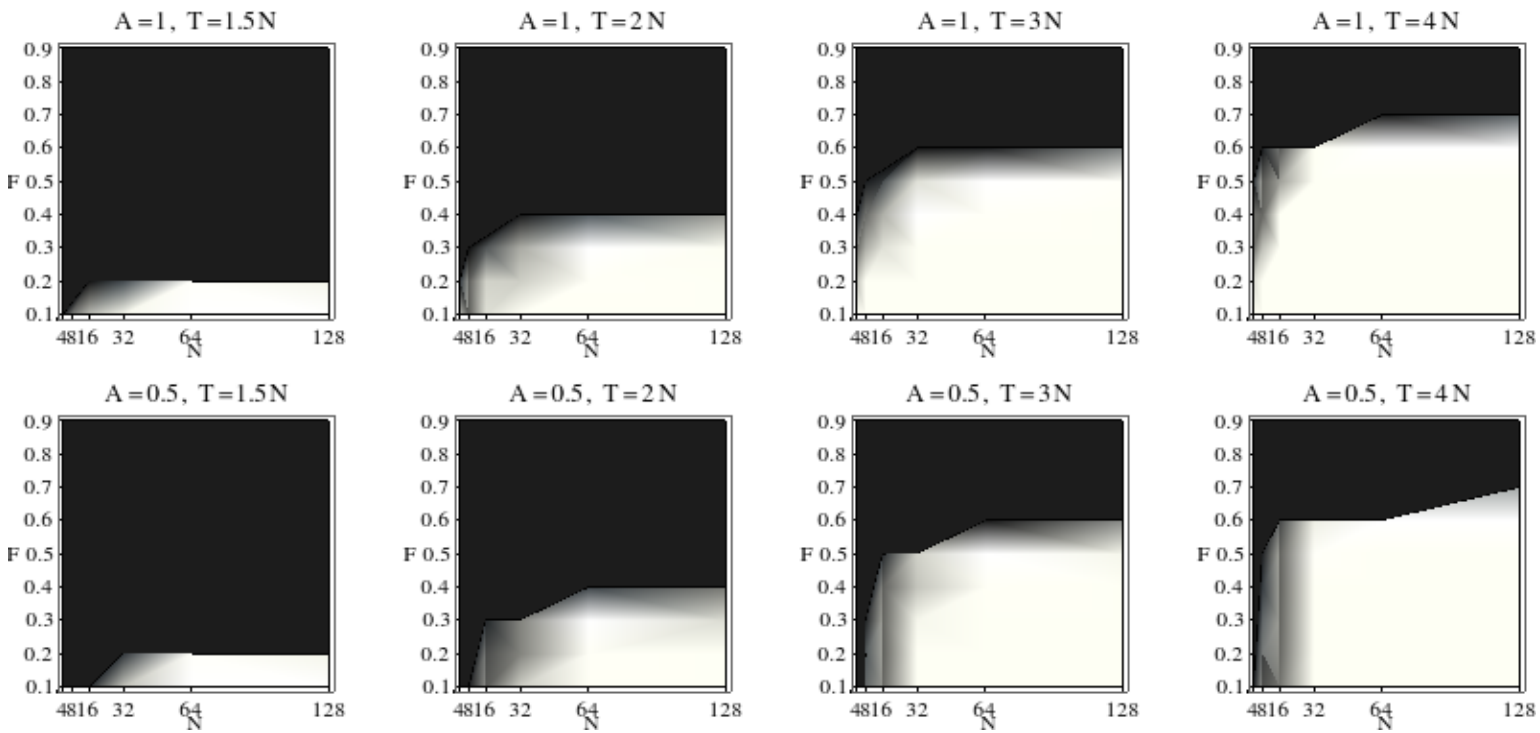


N: number of blocks of a file
A: sparsity factor; ratio of fixed 0 elements (A=1: no sparse coding is used)
R: redundancy; total number of stored blocks

$GF(q) = GF(2^{16})$

Results

Reliability over time, using iterative regeneration

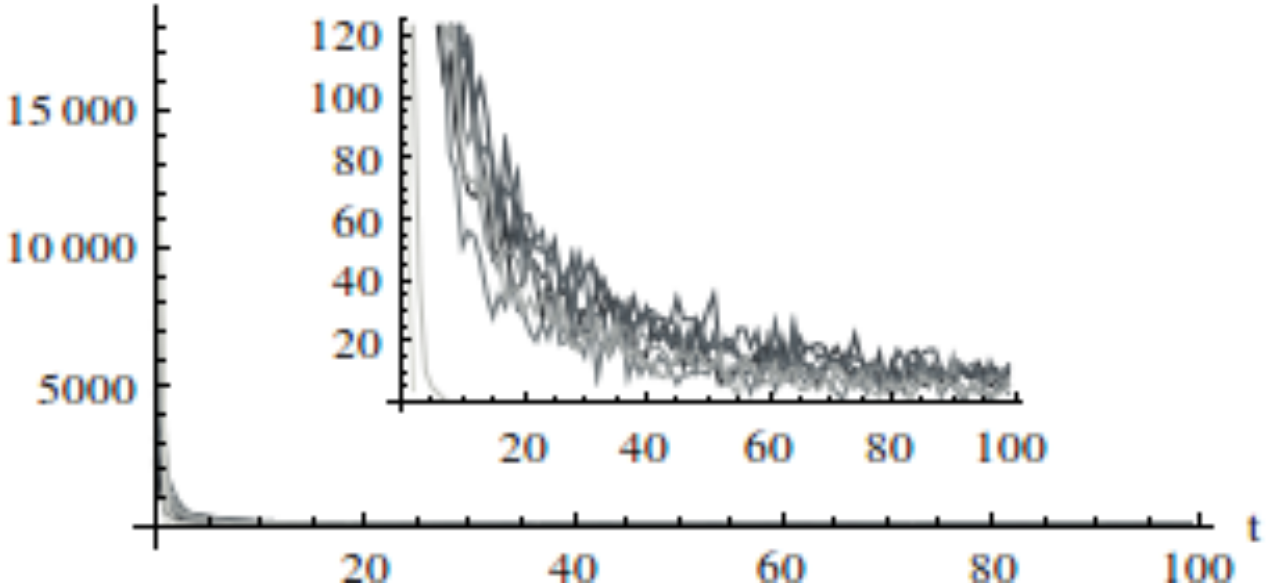


N: number of blocks of a file
A: sparsity factor; ratio of fixed 0 elements (A=1: no sparse coding is used)
T: redundancy; total number of stored blocks
F: ratio of failing nodes between reparations

$GF(q) = GF(2^{16})$

Results

Failure over time



- 90% of all errors happened in the first 15 iterations

$GF(q) = GF(2^{16})$

Conclusions

- Random linear coding: high reliability, near-optimal redundancy
- Iterative reparation
- Sparse coding
  - Decreases CPU requirements
  - Decreases reparation transfer cost
- Non-deterministic
  - Less control overhead
  - Bandwidth overhead is negligible

Thank You!

https://github.com/avisegradi/rnc-lib

Adam Visegradi <adam.visegradi@sztaki.mta.hu>