Coding and computation in distributed storage for dynamic networks

Muriel Médard
Department of EECS
MIT.
Collaborators

- MIT: Supratim Deb (now AT&T research), Tracey Ho (now Speedy Packets), Ben Leong (now National University of Singapore), Prakash Narayana Moorthy, Weifei Zeng

- University of Illinois Urbana-Champaign: Ralf Koetter (later Technical University of Munich)

- University of Aalborg: Frank Fitzek (now Technical University Dresden), Daniel E. Lucani

- Budapest University of Technology and Economics: Hassan Charaf, Marton Sipos, Aron Szabados, Thomas Toth

- Steinwurf: Janus Heide, Morten Pedersen, Peter Vingelmann

- University of Warsaw: Szymon Acedanski.
Overview

• Random linear network coding
• Distributed storage random linear network coding
• Coding in dynamic systems
• Coding for updating functions
Random Linear Network Coding (RLNC)

Algebraic equations more efficiently input data into IP packets

An IP packet payload

Vector of elements of a finite field

4 packets

P1

P2

P3

P4

Random linear combinations are highly likely to be recoverable

P1 + 7P2 + 4P3 + 3P4

2P1 + 8P2 + 9P3 + 5P2

8P1 + 6P2 + P3 + 2P4

7P1 + 2P2 + 3P3 + 4P4

Network Coding and Reliable Communications Group

Coding Algorithm Evolution

Block Codes
- Reed-Solomon (RS)

Convolutional Codes
- 1960s

Modern Codes
- 1990s
  - Low-Density Parity Check Codes (LDPCs)
  - Turbo

Rateless Codes
- 1998
  - Raptor and related codes
  - Rate-less (refinement to End-to-End)
  - Still E2E, still static

Network Codes
- 1998
  - RLNC enables network coding
  - Some special cases allow deterministic codes
    - Index Coding
    - CATWOMAN (Linux 3.10)

Fulcrum Codes
- 2003
  - RLNC-enabled
  - Fluid complexity (flexible field size)
  - Breaks performance-overhead trade-off

- 2014
Commercial Library Benchmarking

• Jerasure 1.2 by James Plank
• Jerasure 2.0 by James Plank
• OpenFEC by INRIA
• ISA-L by INTEL
• KODO by Steinwurf
We have the speed, but that is not our key selling point.

### Comparison with State of the Art

<table>
<thead>
<tr>
<th>Encoding Speed (MB/s)</th>
<th>Generation Size (with Redundancy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>G=8 (12)</td>
<td>Kodo 17 MT (sparse=0.5)</td>
</tr>
<tr>
<td>G=9 (13)</td>
<td>Kodo 17 (sparse=0.5)</td>
</tr>
<tr>
<td>G=10 (15)</td>
<td>ISA-L</td>
</tr>
<tr>
<td>G=16 (24)</td>
<td>Jerasure 2.0</td>
</tr>
<tr>
<td>G=30 (45)</td>
<td>OpenFEC</td>
</tr>
<tr>
<td>G=60 (90)</td>
<td></td>
</tr>
<tr>
<td>G=100 (150)</td>
<td></td>
</tr>
<tr>
<td>G=150 (225)</td>
<td></td>
</tr>
</tbody>
</table>
We have the speed, but that is not our key selling point.

Comparison with State of the Art

Industry Trend

- Benchmarking: Kodo 17 MT (0.5-sparse) vs. best of ISA-L and Jerasure 2.0
- 1MB Packets
- Field Size = $2^8$
- Code Rate = 2/3
Overview

• Random linear network coding
• Distributed storage random linear network coding
• Coding in dynamic systems
• Coding for updating functions
Availability with Coding

Number of peers contacted, one chunk each, to recover the original 10 chunks

Distributed Clouds

- Dropbox
- SkyDrive
- Google Drive
- box
Distributed Clouds

Heterogeneity (4 clouds)

- Clouds behave differently

Speed-Up (5 clouds)
Overview

- Random linear network coding
- Distributed storage random linear network coding
- Coding in dynamic systems
- Coding for updating functions
Dynamic Robustness and Repair

File made up of 16 chunks

Broken into 4 chunk pieces

Stored in 10 data center locations in the cloud

Three of the storage sites are randomly killed per round

How reliably can the data be reconstructed?

Example

File made up of 15 chunks

Stored in 5 racks,
  4 chunks each
Redundancy 33%
File made up of 15 chunks

Stored in 5 racks, 4 chunks each
Redundancy 33%
Reed-Solomon

File made up of 15 chunks

Stored in 5 racks,
4 chunks each
Redundancy 33%
Reed-Solomon

File made up of 15 chunks

Stored in 5 racks, 4 chunks each
Redundancy 33%
Reed-Solomon

File made up of 15 chunks

Stored in 5 racks, 4 chunks each
Redundancy 33%

4 chunks
Reed-Solomon

File made up of 15 chunks

Stored in 5 racks,
4 chunks each
Redundancy 33%
Reed-Solomon

File made up of 15 chunks

Stored in 5 racks,
4 chunks each
Redundancy 33%

Decode
Reed-Solomon

File made up of 15 chunks

Stored in 5 racks,
  4 chunks each
  Redundancy 33%
Reed-Solomon

File made up of 15 chunks

Stored in 5 racks, 4 chunks each
Redundancy 33%

I/O Network: Intra-Rack Inter-Rack Processing
RS: 15 0* 15 Decode + Encode 15x15 matrix (new rack)

RLNC:

* May require some intra-rack transfer depending on structure
RLNC

File made up of 15 chunks

Stored in 5 racks, 4 chunks each
Redundancy 33%

Mix (recode), Send 1 chunk
RLNC

File made up of 15 chunks

Stored in 5 racks, 4 chunks each
Redundancy 33%

Mix (recode), Send 1 chunk
RLNC

File made up of 15 chunks

Stored in 5 racks,
4 chunks each
Redundancy 33%

Mix (recode),
Send 1 chunk
File made up of 15 chunks

Stored in 5 racks, 4 chunks each
Redundancy 33%

Mix (recode), Send 1 chunk
RLNC

File made up of 15 chunks

Stored in 5 racks,
4 chunks each
Redundancy 33%

Racks

Racks

Racks

Racks

Redundancy

Racks

Racks

Racks
RLNC

File made up of 15 chunks

Stored in 5 racks, 4 chunks each
Redundancy 33%

<table>
<thead>
<tr>
<th></th>
<th>I/O</th>
<th>Network: Intra-Rack</th>
<th>Inter-Rack</th>
<th>Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS:</td>
<td>15</td>
<td>0*</td>
<td>15</td>
<td>Decode + Encode 15x15 matrix (new rack)</td>
</tr>
<tr>
<td>RLNC:</td>
<td>15</td>
<td>11</td>
<td>4</td>
<td>Encode 4x4 matrices (4 times), and one 3x3 matrix</td>
</tr>
</tbody>
</table>

* May require some intra-rack transfer depending on structure
RLNC

File made up of 15 chunks

Stored in 5 racks, 4 chunks each
Redundancy 33%

<table>
<thead>
<tr>
<th></th>
<th>I/O</th>
<th>Network: Intra-Rack</th>
<th>Inter-Rack</th>
<th>Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS:</td>
<td>15</td>
<td>0*</td>
<td>15</td>
<td>Centralized in new rack</td>
</tr>
<tr>
<td>RLNC:</td>
<td>15</td>
<td>11</td>
<td>4</td>
<td>Distributed in old and new racks</td>
</tr>
</tbody>
</table>

* May require some intra-rack transfer depending on structure
Memory Consumption RS vs RLNC

Reed-Solomon

RLNC
Using Reed-Solomon leads to larger memory consumption as sufficient segments need to be collected before decoding.

The current example shows three RS processes overlapping in time trying to receive 15 segments before decoding. Inter-arrival process is Poisson-distributed.

Using RLNC leads to lower memory consumption as packets are recoded on arrival such that only one packet is stored per reparation process (independent of the number of segment size).

The current example shows three RLNC processes overlapping in time trying to receive ten segments before decoding. Inter-arrival process in Poisson-distributed.
Memory Consumption RS vs RLNC

Graph showing memory consumption over time for Reed-Solomon (RS) and RLNC codes. The graphs compare the memory consumption of processes starting and ending at specific times, with RS and RLNC codes highlighted in different colors.

- Process 1 starts, Process 2 starts, Process 3 starts
- Process 1 ends, Process 2 ends, Process 3 ends
- Process 1 starts, 2nd packet already recoded, Process 1 ends, Process 1 recoding
Video
Dynamic Robustness and Repair

No Coding

RS Coding

Network Coding

reliability

storage

traffic

reliability

storage

traffic

reliability

storage

traffic
Dynamic Robustness and Repair

No Coding

RS Coding

Network Coding
Overview

- Random linear network coding
- Distributed storage random linear network coding
- Coding in dynamic systems
- Coding for updating functions
Motivation

Decentralized, Distributed Storage Nodes - e.g., P2P or Cloud based

What is the minimum upload needed to update the coded elements?

Current solutions require precise knowledge/tracking of the update vectors
Our solution relies only on estimates of sparsity of the update vectors
What about Computation?

What is the minimum communication necessary for the update?

- Zero probability of error, worst-case scenario
- The function $A$ and sparsity-parameter $k$ are known at the source

P. Narayana Moorthy and Médard, M. “Communication Cost for Updating Functions when Message Updates are Sparse: Connections to Maximally Recoverable Codes”, invited paper, Allerton 2015
Illustrating Matrix for Striped Data File

• E.g. [Length = 5, Dimension = 3] scalar linear code for storage

• \( a_1 = [a_{1,1} \ a_{1,2} \ a_{1,3}] \) coding coefficients for first storage node

\[
\begin{bmatrix}
  a_1 \\
  a_1 \\
  \vdots \\
  a_1
\end{bmatrix}
\begin{bmatrix}
  X_1 \\
  X_2 \\
  X_3 \\
  \vdots \\
  X_{n-2} \\
  X_{n-1} \\
  X_n
\end{bmatrix}
\]

\( m = \text{number of stripes} \)

\( m = 3m \)

1\textsuperscript{st} stripe

\( m\textsuperscript{th} \) stripe
Point-to-Point: Achievable Scheme with $\ell = 2k$

$D$

$H_{2k \times n}(X^n + E^n)$

$S_{2k \times m} H(X^n)$

$H(E^n)$ Syndrome Decoder $\hat{E}^n$

$A(E^n)$

$A(X^n + E^n)$

$A_m \times n(X^n)$

$[n, m] \quad n, 2k$

$[n, 2k]$

$C_A$

$C_H$

1. $H(E^n) = H(\hat{E}^n) \iff A(E^n) = A(\hat{E}^n)$

2. Matrix always exists under sufficiently large field size

$H \quad m > 2k$
Maximally Recoverable Codes: Definition

\[ \mathcal{C}_H \text{ is a Maximally Recoverable Subcode of } \mathcal{C}_A \text{ if } \]

\[ \text{rank}(A|_S) = 2k \implies \text{rank}(H|_S) = 2k, \ \forall S, |S| = 2k \]

\[ A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \]

\[ H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \]
MRCs with Locality in Windows Azure Storage

\[
\begin{bmatrix}
X_1 & X_2 & X_3 & Y_1 & Y_2 & Y_3 & P & Q \\
\end{bmatrix}
\]

\[
C_H : [n = 10, 2k = 6] \text{ code}
\]

Property of MRCs with Locality

• Data decodable from any 6 symbols that are not “dominated” by either of the two local codes
  • E.g. \{X1, Lx, Y1, Ly, P, Q\}
• For this reason, MRCs with locality are better known as Partial MDS codes
  • “as MDS as possible” given the locality constraints
**Point-to-Point : Converse Statements**

\[ X^n + E^n \xrightarrow{H_{\ell \times n}} H(X^n + E^n) \xrightarrow{D} A(X^n + E^n) \]

- \( \ell \geq 2k \) (assuming \( \text{rank}(A) \geq 2k \))

- Under optimality, \( C_H \) must be a \( 2k \) dimensional maximally recoverable subcode of \( C_A \)

Components outside of \( C_A \) are not useful
Broadcast Setting: Problem Statement

What is the minimum communication necessary for updating both destinations simultaneously?
Special Case:

\[ C_A \cap C_B = \{0\} \]

- \( \ell \geq 4k \)
- Optimal to transmit individually to the two destinations – No benefit from broadcasting
Broadcast : Approach for General Case

- Pick \( C_A \cap C_H \ 2k \) as a \( C_A \) - MRSC of
- Pick \( C_B \cap C_H \ 2k \) as a \( C_B \) - MRSC of
- “Maximize” \( C_H \cap C_A \cap C_B \) - we benefit from broadcasting

- Closed form expression for the optimal communication cost can be given
Given $\mathcal{C}_G$ and $\mathcal{C}$, can you construct a maximally recoverable subcode?

- **Necessary Regularity Condition for “sandwiched” MRSC (straightforward):**

$$\text{rank} \left( A \big|_S \right) = t \implies \text{rank} \left( G \big|_S \right) = r, \quad \forall S, |S| = t$$
# Code Constructions

<table>
<thead>
<tr>
<th></th>
<th>Purpose</th>
<th>Field Size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Point-to-Point, $A$ corresponds to stripes of any linear code</td>
<td>$m = \text{rank}(A)$</td>
<td>Partial Maximum Distance Separable codes where local codes are scaled repetition codes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$q^{m-r}$, $A \in F_q^{m \times n}$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Broadcast - “Sandwiched” MRSC, any $A$ and $G$</td>
<td></td>
<td>Based on Linearized Polynomials</td>
</tr>
<tr>
<td>3</td>
<td>A specific family of Partial MDS codes</td>
<td>Better than known constructions</td>
<td>Based on broadcast -“sandwiched” MRSC</td>
</tr>
</tbody>
</table>
Overview

• Random linear network coding
• Distributed storage random linear network coding
• Coding in dynamic systems
• Coding for updating functions
Repair

8 segments (plus redundancy) in 4 clouds

Example: 4 clouds with 3 disks (12 disk storage).

<table>
<thead>
<tr>
<th>Coding Scheme</th>
<th>Disk Storage (less is better)</th>
<th>Inter (Intra) Cloud Bandwidth (less is better)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CLOUD failure</td>
<td>Disk failure</td>
</tr>
<tr>
<td>RS 8:4</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>XORBAS 8:4:2</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0(1)</td>
</tr>
<tr>
<td>RLNC v1a 8:4 systematic</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>RLNC v1b 8:8 systematic</td>
<td>16</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>RLNC v2 dense</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

- Conclusion: RLNC approaches will reduce the traffic at comparable storage situations.
- Staircase/LDPC need significant storage - **unable even to reach 16 in storage**
Reed-Solomon - RS (8,4)

Each storage unit holds some original pieces and a redundancy piece, which is coded from all the original pieces.

Recovery from unit failure:
1. The substitution node receives enough pieces to decode the original data.
2. The data is decoded.
3. The lost redundancy block is encoded.

Recovering from a unit loss requires complete decoding of all data.
XORBAS - like (8,4,3)

Each storage unit holds, in addition to original and redundancy pieces, a local redundancy block.

By adding local redundancy at the cost of additional spent storage, recovery from single block failures requires no transmissions. This “trick” can be applied to other approaches. This enables all units to recover from a single block failure locally, i.e., within the unit. For a unit failure, the cost is the same as for RS.
Perpetual-RLNC (8,4)

Each storage unit holds a perpetually coded block, which is a combination of a subset of the original pieces.

Recovery from unit failure:
1. The remaining units perform recoding to obtain the most useful pieces for the substitution unit.
2. The resulting pieces are transmitted.
3. The lost original pieces are decoded.
4. The lost redundancy block is encoded.

By adding an extra coding step at the sending units, the number of transmissions are reduced and the coding performed at the substitution node simplified.
Perpetual-RLNC* (8,8)

*Random Linear Network Coding

Extra storage can also be spent on decreasing the cost of unit failure repair. Each storage unit holds two perpetually coded blocks. This example considers a smaller subset of original pieces in each coded packet.

Recovery from unit failure:
1. Remaining units perform recoding to obtain pieces for the substitution unit
2. The resulting pieces are transmitted
3. The lost redundancy block is encoded

By utilizing additional storage at each storage unit the number of transmissions can be further reduced.
So far we have considered exact repair if we accept functional repair we can apply RLNC. With RLNC all stored pieces are combinations of all original pieces.

Recovery from unit failure:
1. The remaining units perform uncoordinated recoding combining all pieces they hold.
2. The resulting pieces are transmitted

By utilizing RLNC the number of transmissions is further reduced and the need for coding at the substitution node removed.