On Theory and (Little) Practice of Coding Techniques for Distributed Networked Storage Systems

Frédérique Oggier
Joint work with Anwitaman Datta and Lluís Pàmies-Juárez

Nanyang Technological University, Singapore

Institute of Network Coding, Chinese University of Hong Kong,
January 4 2012
Outline

1. Coding for Distributed Networked Storage
2. Self-Repairing Codes: Constructions and Properties
3. A Little Bit of Practice
Distributed Networked Storage

- A data owner wants to store data over a network of nodes (e.g. data center, back-up or archival in peer-to-peer networks).
- Redundancy is essential for resilience (*Failure is the norm, not the exception*).
- Data from Los Alamos National Laboratory (Dependable Systems and Networks, 2006), gathered over 9 years, 4750 machines and 24101 CPUs. Distribution of failures:
  - Hardware 60%,
  - Software 20%,
  - Network/Environment/Humans 5%,
- Failures occurred between once a day to once a month.
As of June 2011, a study sponsored by the information storage company EMC estimates that the world’s data is more than doubling every 2 years, and reaching 1.8 zettabytes (1 zettabyte = $10^{21}$ bytes) of data to be stored in 2011.¹

If you store this data on DVDs, the stack would reach from the earth to the moon and back.

Redundancy through Coding

- **Replication**: good availability and durability, but very costly.
- **Erasure codes**: good trade-off of availability, durability and storage cost.
Erasure Codes

- A map that takes as input $k$ blocks of data and outputs $n$ blocks of data, $n - k$ of them thus giving redundancy.
- An $(n, k)$ erasure code is characterized by (1) how many blocks are needed to decode (recover) the $k$ blocks of original data - if any choice of $k$ encoded blocks can do, the code is called maximum distance separable (MDS) and (2) its rate $k/n$ (or storage overhead $n/k$).
- 3 way replication is a (3,1) erasure code.
Erasure codes for communication

$k$ blocks

$n$ encoded blocks
(to be sent through an “Erasure” communication channel)

Receive any $k'$ (≥ $k$) blocks

Lost blocks

Original $k$ blocks

Reconstruct Data
Erasure codes for storage systems

- Data = Object
- Encoding
- Decoding
- Reconstruct Data

$k$ blocks

$n$ encoded blocks (stored in storage devices in a network)

Retrieve any $k' (\geq k)$ blocks

Lost blocks

$O_k$
Nodes may go offline, or may fail, so that the data they store becomes *unavailable*.

Redundancy needs to be *replenished*, else data may be permanently lost over time (after multiple storage node failures).
Repair process using traditional Erasure Codes

Retrieve any \( k' \geq k \) blocks

Lost blocks

\( n \) encoded blocks
(stored in storage devices in a network)

Decoding

\( B_1 \)
\( B_2 \)
\( \vdots \)
\( B_n \)

\( O_1 \)
\( O_2 \)
\( \vdots \)
\( O_k \)

 Encoding
Recreate lost blocks

\( B_l \)

 Re-insert

Reinsert in (new) storage devices, so that there is (again) \( n \) encoded blocks
Related work


Regenerating Codes

- Based on Network Coding (max flow-min cut argument) on top of an MDS \((n, k)\) erasure code.
- Characterize storage overhead - repair bandwidth trade-off.
- Number of contacted live nodes to repair is at least \(k\).
Collaborative Regenerating Codes

- Allow collaboration among new comers.
- Improve the storage overhead - repair bandwidth trade-off.
- Tolerates multiple faults.
Codes for Storage: Wish List

- Low storage overhead,
- Good fault tolerance,
- Low repair bandwidth cost,
- Low repair time,
- Low complexity,
- I/O
- ...
Outline

1. Coding for Distributed Networked Storage
2. Self-Repairing Codes: Constructions and Properties
3. A Little Bit of Practice
Self-Repairing Codes (SRC)

- **Motivation**: *minimize* the number of nodes necessary to repair a missing block.
  - The minimum is 2, cannot be achieved without sacrificing the MDS property.

- **Self-repairing codes** are \((n, k)\) codes such that
  - encoded fragments can be repaired *directly* from other subsets of encoded fragments,
  - a fragment can be repaired from a **fixed number** of encoded fragments (typically 2 or 3), *independently* of which specific blocks are missing (analogous to erasure codes supporting reconstruction using any \(n - k\) losses, independently of which).
Self-Repairing Codes (a black-box view)

Retrieve some \( k'' (< k) \) blocks (e.g. \( k'' = 2 \)) to recreate a lost block.

Lost blocks

\[ B_1, B_2, \ldots, B_n \]

\( n \) encoded blocks
(stored in storage devices in a network)

Reinsert in (new) storage devices, so that there is (again) \( n \) encoded blocks.
Homomorphic SRC (HSRC)

- A first instance of self-repairing code.
- Based on polynomial evaluation.
- An object is cut into $k$ pieces, which represent coefficients of a polynomial $p$. The $k$ pieces are mapped to $n$ encoded fragments, by performing $n$ polynomial evaluations ($p(\alpha_1), \ldots, p(\alpha_n)$).

Self-repairing Homomorphic Codes for Distributed Storage Systems
F. Oggier, A. Datta, *INFOCOM 2011*
HSRC: Encoding Illustration

Data = Object

\[ p(X) = \sum_{i=0}^{k-1} p_i X^{2i} \]

with \( p_i = O_{i+1} \)

Linearized polynomial

\( k \) blocks

Each of size \( M/k \)

\( n \) encoded blocks

\( B_1 \)
\( B_2 \)
\( B_3 \)
\( \ldots \)
\( B_n \)

\( p(\alpha_1) \)
\( p(\alpha_2) \)
\( \ldots \)
\( p(\alpha_n) \)
HSRC: Decoding and Repair

1. **Decoding** is ensured by Lagrange interpolation.
2. **Repair**: \( p(a + b) = p(a) + p(b) \).
3. Computational cost of a repair: XORs.
HSRC: A toy example

- Cut a file into $k = 3$ fragments, which serve as coefficients for a polynomial $p$.
- For $n = 7$, evaluate $p(X)$ at say $1, w, w^2, w^4, w^5, w^8, w^{10}$. We get:

$$\left(p(1), p(w), p(w^2), p(w^4), p(w^5), p(w^8), p(w^{10})\right)$$

<table>
<thead>
<tr>
<th>missing fragment(s)</th>
<th>pairs to reconstruct missing fragment(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(1)$</td>
<td>$(p(w), p(w^4));(p(w^2), p(w^8));(p(w^5), p(w^{10}))$</td>
</tr>
<tr>
<td>$p(w)$</td>
<td>$(p(1), p(w^4));(p(w^2), p(w^5));(p(w^8), p(w^{10}))$</td>
</tr>
<tr>
<td>$p(w^2)$</td>
<td>$(p(1), p(w^8));(p(w), p(w^5));(p(w^4), p(w^{10}))$</td>
</tr>
<tr>
<td>$p(1)$ and $p(w)$</td>
<td>$(p(w^2), p(w^8))$ or $(p(w^5), p(w^{10}))$ for $p(1)$</td>
</tr>
<tr>
<td>$p(1)$ and $p(w)$</td>
<td>$(p(w^8), p(w^{10}))$ or $(p(w^2), p(w^5))$ for $p(w)$</td>
</tr>
<tr>
<td>$p(1)$ and $p(w)$</td>
<td>$(p(w^5), p(w^{10}))$ for $p(1)$</td>
</tr>
<tr>
<td>$p(1)$ and $p(w)$</td>
<td>$(p(w^8), p(w^{10}))$ for $p(w)$</td>
</tr>
<tr>
<td>$p(1)$ and $p(w)$</td>
<td>$(p(w^4), p(w^{10}))$ for $p(w^2)$</td>
</tr>
</tbody>
</table>
Self-Repairing Codes from Projective Geometry (PSRC)

- A second instance of self-repairing code, based on spreads.
- Spread = partition of the space into subspaces, nodes store inner product of the data with basis vectors of subspaces.

Self-Repairing Codes for Distributed Storage - A Projective Geometric Construction, F. Oggier, A. Datta, ITW 2011
### PSRC: A toy example

<table>
<thead>
<tr>
<th>node</th>
<th>basis vectors</th>
<th>data stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_1$</td>
<td>$v_1 = (1000)$, $v_2 = (0110)$</td>
<td>${o_1, o_2 + o_3}$</td>
</tr>
<tr>
<td>$N_2$</td>
<td>$v_3 = (0100)$, $v_4 = (0011)$</td>
<td>${o_2, o_3 + o_4}$</td>
</tr>
<tr>
<td>$N_3$</td>
<td>$v_5 = (0010)$, $v_6 = (1101)$</td>
<td>${o_3, o_1 + o_2 + o_4}$</td>
</tr>
<tr>
<td>$N_4$</td>
<td>$v_7 = (0001)$, $v_8 = (1010)$</td>
<td>${o_4, o_1 + o_3}$</td>
</tr>
<tr>
<td>$N_5$</td>
<td>$v_9 = (1000)$, $v_{10} = (0101)$</td>
<td>${o_1 + o_2, o_2 + o_4}$</td>
</tr>
</tbody>
</table>
Static resilience

- There is at least one pair to repair a node, for up to \((n - 1)/2\) simultaneous failures.
- **Static resilience** of a distributed storage system is the probability that an object stored in the system stays available without any further maintenance, even when a fraction of nodes become unavailable.
Static resilience: HSRC versus EC

Figure: Static resilience of self-repairing codes (SRC): Validation of analysis, and comparison with erasure codes (EC)
Static resilience: PSRC versus EC

Figure: Static resilience of self-repairing codes (HSRC): Comparison with erasure codes (EC)
More on Resilience: HSRC versus EC

Figure: Static resilience of self-repairing codes (HSRC) versus erasure codes (EC)
More on Resilience: PSRC versus EC

Figure: Static resilience of self-repairing codes (HSRC): Comparison with erasure codes (EC)

F. Oggier (NTU)
Fast & parallel repairs using HSRC: A toy example

- Consider:
  - (15,4) code, nodes storing $p(w^i)$ for $i = 0, 1, 2, 3, 4, 5, 6$ are missing
  - Nodes have upload/download bandwidth limit: one block per time unit
- Possible pairs to repair each missing block:

<table>
<thead>
<tr>
<th>fragment</th>
<th>suitable pairs to reconstruct</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p(1)$</td>
<td>$(p(w^1), p(w^9)); (p(w^{11}), p(w^{12}))$</td>
</tr>
<tr>
<td>$p(w)$</td>
<td>$(p(w^7), p(w^{14})); (p(w^8), p(w^{10}))$</td>
</tr>
<tr>
<td>$p(w^2)$</td>
<td>$(p(w^7), p(w^{12})); (p(w^9), p(w^{11})); (p(w^{12}), p(w^{10}))$</td>
</tr>
<tr>
<td>$p(w^3)$</td>
<td>$(p(w^8), p(w^{13})); (p(w^{10}), p(w^{12}))$</td>
</tr>
<tr>
<td>$p(w^4)$</td>
<td>$(p(w^9), p(w^{14})); (p(w^{11}), p(w^{13}))$</td>
</tr>
<tr>
<td>$p(w^5)$</td>
<td>$(p(w^7), p(w^{13})); (p(w^{12}), p(w^{14}))$</td>
</tr>
<tr>
<td>$p(w^6)$</td>
<td>$(p(w^7), p(w^{10})); (p(w^8), p(w^{14}))$</td>
</tr>
</tbody>
</table>

- A parallelized schedule:

<table>
<thead>
<tr>
<th>node</th>
<th>$p(w^0)$</th>
<th>$p(w^1)$</th>
<th>$p(w^2)$</th>
<th>$p(w^3)$</th>
<th>$p(w^4)$</th>
<th>$p(w^5)$</th>
<th>$p(w^6)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time 1</td>
<td>$p(w^7)$</td>
<td>$p(w^8)$</td>
<td>$p(w^9)$</td>
<td>$p(w^{13})$</td>
<td>$p(w^{11})$</td>
<td>$p(w^{12})$</td>
<td>$p(w^{10})$</td>
</tr>
<tr>
<td>Time 2</td>
<td>$p(w^9)$</td>
<td>$p(w^{10})$</td>
<td>$p(w^11)$</td>
<td>$p(w^8)$</td>
<td>$p(w^{13})$</td>
<td>$p(w^{14})$</td>
<td>$p(w^7)$</td>
</tr>
</tbody>
</table>
## Systematic Object Retrieval using PSRC: A toy example

<table>
<thead>
<tr>
<th>node</th>
<th>basis vectors</th>
<th>data stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_1$</td>
<td>$v_1 = (1000)$, $v_2 = (0110)$</td>
<td>${o_1, o_2 + o_3}$</td>
</tr>
<tr>
<td>$N_2$</td>
<td>$v_3 = (0100)$, $v_4 = (0011)$</td>
<td>${o_2, o_3 + o_4}$</td>
</tr>
<tr>
<td>$N_3$</td>
<td>$v_5 = (0010)$, $v_6 = (1101)$</td>
<td>${o_3, o_1 + o_2 + o_4}$</td>
</tr>
<tr>
<td>$N_4$</td>
<td>$v_7 = (0001)$, $v_8 = (1010)$</td>
<td>${o_4, o_1 + o_3}$</td>
</tr>
<tr>
<td>$N_5$</td>
<td>$v_9 = (1100)$, $v_{10} = (0101)$</td>
<td>${o_1 + o_2, o_2 + o_4}$</td>
</tr>
</tbody>
</table>
Outline

1. Coding for Distributed Networked Storage

2. Self-Repairing Codes: Constructions and Properties

3. A Little Bit of Practice
A Little Bit of Practice

A More Realistic Scenario

- A network with 1000 (full duplex) nodes,
- 10,000 objects of size 1GB are stored,
- Multiple failures.

*Pipelined codes* are also considered.
Simulation Results: Storage of Multiple Objects

Figure 4: Analysis of the system performance using different codes when a fraction $\Theta$ of nodes fails simultaneously. The size of the objects is $B = 1$GB, $L = 10,000$ objects are randomly stored in $N = 1,000$ nodes.
Data Insertion

- Replication: To store a new object, a source node uploads one replica to a first node, which can concurrently forward it to another storage node, etc.
- Erasure Codes: The source node computes and uploads the encoded fragments to the corresponding storage nodes.
- **Issue**: insertion time, possibly worsened by mismatched temporal constraints (e.g. F2F).

*In-Network Redundancy Generation for Opportunistic Speedup of Backup*, L. Pamies-Juarez, A. Datta, F. Oggier *preprint*
Simulation Results: In-Network Coding

- IM traces for F2F scenario
- Figure (1): storage throughput increases with node availability. Figure (2): total traffic increases, scales with storage throughput. Figure (3): reduction of data upload at the source, up to 40%.
Future/ongoing work

- Efficient decoding, other instances of SRC
- Implementation & integration in a distributed storage system
- Various systems/algorithmic issues: Topology optimized placement, repair scheduling
Q&A

- More information: http://sands.sce.ntu.edu.sg/CodingForNetworkedStorage/
- Contact: \{frederique, anwitaman\}@ntu.edu.sg