On Coding Techniques for Networked Distributed Storage Systems

Frédérique Oggier
frederique@ntu.edu.sg

Nanyang Technological University, Singapore

First European Training School on Network Coding, Barcelona, February 2013
Outline

1. Coding for Distributed Networked Storage
2. Regenerating Codes
3. Locally Repairable Codes
4. Data Insertion / Migration
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.
What’s New: More Numbers

- 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.

  ![The Economist](http://www.emc.com/about/news/press/2011/20110628-01.htm)

- 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

- **Data**: $O_1, O_2, \ldots, O_k$
- **Encoded Blocks**: $B_1, B_2, \ldots, B_n$
- **Received Blocks**: $O_1, O_2, \ldots, O_k$

**Steps**:
1. Encoding
2. Transmission
3. Decoding

**Key Points**:
- **$k$ blocks**
- **$n$ encoded blocks**
  (to be sent through an “Erasure” communication channel)
- **Receive any $k' \geq k$ blocks**
- **Reconstruct Data**
Erasure codes for storage systems

- **Data = Object**
  - $O_1$
  - $O_2$
  - $O_k$

- **Encoding**
  - $B_1$
  - $B_2$
  - $B_i$
  - $B_n$

- **Decoding**
  - Retrieve any $k'$ ($\geq k$) blocks

- **Reconstruct Data**
  - $O_1$
  - $O_2$
  - $O_k$

- **Initial Blocks**
  - $k$ blocks

- **Encoded Blocks**
  - $n$ encoded blocks
  - (stored in storage devices in a network)
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

Decoding

Recalculate lost blocks

Encoding

Reinsert lost blocks in (new) storage devices, so that there is (again) $n$ encoded blocks

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


8. K. W. Shum, *Cooperative regenerating codes for distributed storage systems Communications*, ICC 2011.
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.

[K. Shum, *Cooperative Regenerating Codes for Distributed Storage Systems, ICC 2011*],
[Kermarrec, Le Scouarnec, Straub, *Repairing multiple failures with coordinated and adaptive regenerating codes, Netcod 2011*]
Exact vs Functional Repair

- **Exact Repair**: the repaired data is bit-by-bit identical to the lost one.
- **Functional Repair**: the repaired data is such that the information about the stored object is preserved.

  - **Storage code**: a collection of $n$ subspaces $U_1, \ldots, U_n$ of an $m$-dimensional vector space $U$ over some finite field $\mathbb{F}_q$, each of dimension $\alpha$.
  - **Data recovery**: there must exist subsets of these $U_i$ such that they span the entire vector space $U$.
  - **Data repair**: a subset $U_l$ can be repaired from a collection of $\beta$-dimensional subspaces $W_{i,l} \subset U_i$ if $U_l$ is contained in the span of the $W_{i,l}$.

[Hollmann, *Storage codes - coding rate and repair locality*, ICNC 2013]
Codes for Storage: Wish List

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

F. Oggier (NTU)

Coding for Storage
Outline

1. Coding for Distributed Networked Storage
2. Regenerating Codes
3. Locally Repairable Codes
4. Data Insertion / Migration
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 (when minimum of data is stored).
- **Self-repairing codes** are \((n, k)\) codes such that a fragment can be repaired from a **fixed number** of encoded fragments (typically 2 or 3), **independently** of which specific blocks are missing.
Self-Repairing Codes (a black-box view)

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

Lost blocks

$n$ 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 evaluation of linearized polynomials.
- 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 ($p(\alpha_1), \ldots, p(\alpha_n)$).

Whenever the number of nodes contacted for repair is less than $k$, where $k$ is the number of nodes needed to recover the data.

[P. Gopalan, C. Huang, H. Simitci, S. Yekhanin, *On the Locality of Codeword Symbols*]
[D.S. Papailiopoulos and A.G. Dimakis, *Locally Repairable codes, ISIT 2012*]
Object $o = (o_1, o_2, o_3, o_4)$ to be stored.

<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>

A Toy Example (of Self-Repairing Code) II

<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>

**Repair using two nodes**

Say $N_1$ and $N_3$

$\{o_1 + o_2 + o_4\} + (o_1) \Rightarrow o_2 + o_4$

$\{o_3\} + (o_2 + o_3) \Rightarrow o_2$

$(o_1) + (o_2) \Rightarrow o_1 + o_2$

Four pieces needed to regenerate two pieces

**Repair using three nodes**

Say $N_2$, $N_3$ and $N_4$

$\{o_2\} + (o_4) \Rightarrow o_2 + o_4$

$(o_1 + o_2 + o_4) + (o_4) \Rightarrow o_1 + o_2$

Three pieces needed to regenerate two pieces
Static resilience

- **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.
### Systematic Object Retrieval

<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>
Summary on Repairability

- Minimize the amount of repair bandwidth (e.g. Regenerating Codes).
- Minimize the amount of contacted nodes during repair (e.g. Self-Repairing and generally Locally Repairable Codes), which in term might reduce the amount of bandwidth as well.
- Minimize the repair time.
- Optimize the number of repair options.
So Far: Redundancy (Reliability/Storage Overhead)
Locally Repairable Codes

So Far: Repair

Replicas

Erasure coded data

(local)

repair

Data access
Outline

1. Coding for Distributed Networked Storage
2. Regenerating Codes
3. Locally Repairable Codes
4. Data Insertion / Migration
Data Insertion

Data insertion

Replicas

Erasure coded data

(local) repair

pipelined insertion

repair

e.g., data for analytics

In-network coding

e.g., multimedia

Data access
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).

[L. Pamies-Juarez, A. Datta, F. Oggier, *In-Network Redundancy Generation for Opportunistic Speedup of Backup*]
In-Network Coding

- Exploits the local repairability.
- Different scheduling strategies were studied: increase in the data insertion throughput, up to 40% (at the cost of 20% increase in network traffic).
Data Migration

- Data insertion
- Pipelined insertion, e.g., data for analytics
- In-network coding, e.g., multimedia
- Erasure coded data
- Replicas
- Archival of "cold" data
RapidRAID: Motivation

- Archival of cold data.
- Implement a decentralized encoding process.

Fig. 1. Network flow required to encode a data object using a classical systematic (8,4) erasure code using nodes n1...n8. The ⊕ symbol denotes a coding operation.

Fig. 2. Network flow required to encode a data object using a (8,4) pipelined erasure code using node n1...n8. The ⊕ symbol denotes a coding operation.
RapidRAID: Coding

- Encoding is summarized by a generator matrix.
- Reduce a single object’s coding time by up to 90%, while when multiple objects are encoded concurrently, the reduction is up to 20%.

\[
\begin{align*}
c_1 &= o_1 \xi_1, \\
c_2 &= x_{1,2} + o_2 \xi_2 = o_1 \psi_1 + o_2 \xi_2, \\
c_3 &= x_{2,3} + o_3 \xi_3 = o_1 \psi_1 + o_2 \psi_2 + o_3 \xi_3, \\
c_4 &= x_{3,4} + o_4 \xi_4 = o_1 \psi_1 + o_2 \psi_2 + o_3 \psi_3 + o_4 \xi_4, \\
c_5 &= x_{4,5} + o_1 \xi_5 \\
&= o_1(\psi_1 + \xi_5) + o_2 \psi_2 + o_3 \psi_3 + o_4 \psi_4, \\
c_6 &= x_{5,6} + o_2 \xi_6 \\
&= o_1(\psi_1 + \psi_5) + o_2(\psi_2 + \xi_6) + o_3 \psi_3 + o_4 \psi_4, \\
c_7 &= x_{6,7} + o_3 \xi_7 \\
&= o_1(\psi_1 + \psi_5) + o_2(\psi_2 + \psi_6) + o_3(\psi_3 + \xi_7) + o_4 \psi_4, \\
c_8 &= x_{7,8} + o_4 \xi_8 \\
&= o_1(\psi_1 + \psi_5) + o_2(\psi_2 + \psi_6) + o_3(\psi_3 + \psi_7) + o_4(\psi_4 + \xi_8),
\end{align*}
\]

\[
\begin{bmatrix}
\xi_1 \\
\psi_1 \\
\psi_1 \\
\psi_1 + \xi_5 \\
\psi_1 + \psi_5 \\
\psi_1 + \psi_5 \\
\psi_1 + \psi_5 \\
\psi_1 + \psi_5
\end{bmatrix}
\begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
o_1 \\
o_2 \\
o_3 \\
o_4
\end{bmatrix}
= 
\begin{bmatrix}
c_1 \\
c_2 \\
c_3 \\
c_4 \\
c_5 \\
c_6 \\
c_7 \\
c_8
\end{bmatrix}
\]
Summary

- Erasure Codes for Communication vs for Networked Distributed Storage Systems
- Fault tolerance vs Storage Overhead
- Repairability: Repair Bandwidth, Local Repair, Repair Time
- Data Insertion and Migration to archival.
Future/ongoing work

- Efficient decoding
- Implementation & integration in a distributed storage system
- Various systems/algorithmic issues: Topology optimized placement, repair scheduling

and design of new codes...
Q&A

- More information:
  http://sands.sce.ntu.edu.sg/CodingForNetworkedStorage/
- Advertisement: a short and a long survey are available.