Storage codes: Managing Big Data with Small Overheads

Presented by

Anwitaman Datta & Frédérique E. Oggier
Nanyang Technological University, Singapore

Tutorial at NetCod 2013, Calgary, Canada.
A note from the trenches: "You know you have a large storage system when you get paged at 1 AM because you only have a few petabytes of storage left." – from Andrew Fikes’ (Principal Engineer, Google) faculty summit talk `Storage Architecture and Challenges`, 2010.
Big Data Storage: Disclaimer

A note from the trenches: "You know you have a large storage system when you get paged at 1 AM because you only have a few petabytes of storage left." — Andrew Fikes’ (Principal Engineer, Google) faculty summit talk `Storage Architecture and Challenges `, 2010.

We never get such calls!!
Big data

• June 2011 EMC² study
  – world’s data is *more than doubling* every 2 years
    • faster than Moore’s Law
  – 1.8 zettabytes of data to be created in 2011

Zetta: $10^{21}$

Zettabyte: If you stored all of this data on DVDs, the stack would reach from the Earth to the moon and back.

The data deluge: Some numbers

• **Facebook** “currently” (in 2010) stores over 260 billion images, which translates to over 20 petabytes of data. Users upload one billion new photos (60 terabytes) each week and Facebook serves over one million images per second at peak. [quoted from a paper on “Haystack” from Facebook]

• On “Saturday”, **photo number four billion** was uploaded to photo sharing site [Flickr](http://mashable.com/2009/10/12/flickr-4-billion/). This comes just five and a half months after the 3 billionth and nearly 18 months after photo number two billion. – Mashable (13th October 2009)

© 2013, A. Datta & F. Oggier, NTU Singapore
Scale how?
Scale how?

To scale vertically (or **scale up**) means to add resources to a single node in a system*

* Definitions from Wikipedia
Scale how?

not distributing is not an option!

To scale vertically (or **scale up**) means to add resources to a single node in a system*

To scale horizontally (or **scale out**) means to add more nodes to a system, such as adding a new computer to a distributed software application*

* Definitions from Wikipedia
Failure (of parts) is Inevitable
Failure (of parts) is Inevitable

• But, failure of the system is not an option either!
  – Failure is the pillar of rivals’ success …
Deal with it

- Data from Los Alamos National Laboratory (DSN 2006), gathered over 9 years, 4750 machines, 24101 CPUs.
- Distribution of failures:
  - Hardware 60%
  - Software 20%
  - Network/Environment/Humans 5%
- Failures occurred between once a day to once a month.
Failure happens without fail, so …

• But, failure of the system is not an option either!
  – Failure is the pillar of rivals’ success …

• Solution: Redundancy
Many Levels of Redundancy

- Physical
- Virtual resource
- Availability zone
- Region
- Cloud

Redundancy Based Fault Tolerance

- Replicate data
  - e.g., 3 or more copies
  - In nodes on different racks
    - Can deal with switch failures
- Power back-up using battery between racks (Google)
But At What Cost?

• Failure is not an option, but …
  – … are the overheads acceptable?
Reducing the Overheads of Redundancy

• Erasure codes
  – Much lower storage overhead
  – High level of fault-tolerance
    • In contrast to replication or RAID based systems
• Has the potential to significantly improve the “bottomline”
  – e.g., Both Google’s new DFS Colossus, as well as Microsoft’s Azure now use ECs
Erasure Codes (ECs)
Erasure Codes (ECs)

- An \((n,k)\) erasure code = a map that takes as input \(k\) blocks and outputs \(n\) blocks, thus introducing \(n-k\) blocks of redundancy.
Erasure Codes (ECs)

• An \((n,k)\) erasure code = a map that takes as input \(k\) blocks and outputs \(n\) blocks, thus introducing \(n-k\) blocks of redundancy.

• 3 way replication is a \((3,1)\) erasure code!
Erasure Codes (ECs)

- An \((n,k)\) erasure code = a map that takes as input \(k\) blocks and outputs \(n\) blocks, thus introducing \(n-k\) blocks of redundancy.
- 3 way replication is a \((3,1)\) erasure code!

\(k=1\) block
Erasure Codes (ECs)

• An \((n,k)\) erasure code = a map that takes as input \(k\) blocks and outputs \(n\) blocks, thus introducing \(n-k\) blocks of redundancy.
• 3 way replication is a \((3,1)\) erasure code!

\(k=1\) block \hspace{1cm} n=3\text{ encoded blocks}
Erasure Codes (ECs)

- An (n,k) erasure code = a map that takes as input k blocks and outputs n blocks, thus introducing n-k blocks of redundancy.
- 3 way replication is a (3,1) erasure code!

An erasure code such that the k original blocks can be recreated out of any k encoded blocks is called **MDS** (maximum distance separable).
Erasure Codes (ECs)

- Originally designed for communication
  - EC\((n,k)\)

Data = message

Encoding

\(k\) blocks

\(n\) encoded blocks

Decoding

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

Lost blocks

Reconstruct Data

Original \(k\) blocks

\(O_1\)

\(O_2\)

\(O_k\)

\(B_1\)

\(B_2\)

\(B_n\)
Erasure Codes for Networked Storage

$k$ blocks

$n$ encoded blocks

(stored in storage devices in a network)

Data = Object

Encoding

Retrieval of $k'$ ($\geq k$) blocks

Lost blocks

Decoding

Reconstruct Data

© 2013, A. Datta & F. Oggier, NTU Singapore
HDFS-RAID

• Distributed RAID File system (DRFS) client
  – provides application access to the files in the DRFS
  – transparently recovers any corrupt or missing blocks encountered when reading a file (degraded read)
  • Does not carry out repairs

• RaidNode, a daemon that creates and maintains parity files for all data files stored in the DRFS

• BlockFixer, which periodically recomputes blocks that have been lost or corrupted
  – RaidShell allows on demand repair triggered by administrator

• Two kinds of erasure codes implemented
  – XOR code and Reed-Solomon code (typically 10+4 w/ 1.4x overhead)

From http://wiki.apache.org/hadoop/HDFS-RAID

© 2013, A. Datta & F. Oggier, NTU Singapore
Replenishing Lost Redundancy for ECs

- Repair needed for long term resilience.

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

Decoding

Recreate lost blocks

Encoding

Re-insert

Lost blocks

Original $k$ blocks

$n$ encoded blocks
Replenishing Lost Redundancy for ECs

• Repair needed for long term resilience.

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

Recreate lost blocks

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

• Repairs are expensive!
Tailored-Made Codes for Storage
Tailored-Made Codes for Storage

Desired code properties include:

- Low storage overhead
- Good fault tolerance

Traditional MDS erasure codes achieve these.
Tailored-Made Codes for Storage

Desired code properties include:

- Low storage overhead
- Good fault tolerance
- Better repairability

Traditional MDS erasure codes achieve these.
Tailored-Made Codes for Storage

Desired code properties include:

- Low storage overhead
- Good fault tolerance
- Better repairability

Traditional MDS erasure codes achieve these.

- Smaller repair fan-in
- Reduced I/O for repairs
- Possibility of multiple simultaneous repairs
- Fast repairs
- Efficient B/W usage
- ...
Tailored-Made Codes for Storage

Desired code properties include:
- Low storage overhead
- Good fault tolerance
- Better repairability
- Better …

Traditional MDS erasure codes achieve these.
- Smaller repair fan-in
- Reduced I/O for repairs
- Possibility of multiple simultaneous repairs
- Fast repairs
- Efficient B/W usage
- …
Tailored-Made Codes for Storage

Desired code properties include:

- Low storage overhead
- Good fault tolerance
- Better repairability
- Better repairability

Traditional MDS erasure codes achieve these.

- Smaller repair fan-in
- Reduced I/O for repairs
- Possibility of multiple simultaneous repairs
- Fast repairs
- Efficient B/W usage
- ...

- Better data-insertion
- Better migration to archival
- ...

© 2013, A. Datta & F. Oggier, NTU Singapore
Pyramid (Local Reconstruction) Codes

Pyramid Codes: Flexible Schemes to Trade Space for Access Efficiency in Reliable Data Storage Systems, C. Huang et al. @ NCA 2007

Erasure Coding in Windows Azure Storage, C. Huang et al. @ USENIX ATC 2012
Pyramid (Local Reconstruction) Codes

- Good for degraded reads (data locality)

Pyramid Codes: Flexible Schemes to Trade Space for Access Efficiency in Reliable Data Storage Systems, C. Huang et al. @ NCA 2007

Erasure Coding in Windows Azure Storage, C. Huang et al. @ USENIX ATC 2012

© 2013, A. Datta & F. Oggier, NTU Singapore
Pyramid (Local Reconstruction) Codes

- Good for degraded reads (data locality)
- Not all repairs are cheap (only partial parity locality)

Pyramid Codes: Flexible Schemes to Trade Space for Access Efficiency in Reliable Data Storage Systems, C. Huang et al. @ NCA 2007

Erasure Coding in Windows Azure Storage, C. Huang et al. @ USENIX ATC 2012
Regenerating Codes

• Network information flow based arguments to determine “optimal” trade-off of storage/repair-bandwidth
Locally Repairable Codes

- Codes satisfying: low repair fan-in, for any failure
- The name is reminiscent of “locally decodable codes”
Self-repairing Codes
Self-repairing Codes

• Usual disclaimer: “To the best of our knowledge”
  – First instances of locally repairable codes
    • Self-repairing Homomorphic Codes for Distributed Storage Systems
      – Infocom 2011
    • Self-repairing Codes for Distributed Storage Systems – A Projective Geometric Construction
      – ITW 2011
  – Since then, there have been many other instances from other researchers/groups
Self-repairing Codes

• Usual disclaimer: “To the best of our knowledge”
  – First instances of locally repairable codes
    • Self-repairing Homomorphic Codes for Distributed Storage Systems
      – Infocom 2011
    • Self-repairing Codes for Distributed Storage Systems – A Projective Geometric Construction
      – ITW 2011
  – Since then, there have been many other instances from other researchers/groups

• Note
  – k encoded blocks are enough to recreate the object
    • Caveat: not any arbitrary k (i.e., SRCs are not MDS)
    • However, there are many such k combinations
Self-repairing Codes: Blackbox 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
## PSRC 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>
## PSRC 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>
### PSRC Example

<table>
<thead>
<tr>
<th>node</th>
<th>basis vectors</th>
<th>data stored</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_1$</td>
<td>$\nu_1 = (1000)$, $\nu_2 = (0110)$</td>
<td>${o_1, o_2 + o_3}$</td>
</tr>
<tr>
<td>$N_2$</td>
<td>$\nu_3 = (0100)$, $\nu_4 = (0011)$</td>
<td>${o_2, o_3 + o_4}$</td>
</tr>
<tr>
<td>$N_3$</td>
<td>$\nu_5 = (0010)$, $\nu_6 = (1101)$</td>
<td>${o_3, o_1 + o_2 + o_4}$</td>
</tr>
<tr>
<td>$N_4$</td>
<td>$\nu_7 = (0001)$, $\nu_8 = (1010)$</td>
<td>${o_4, o_1 + o_3}$</td>
</tr>
<tr>
<td>$N_5$</td>
<td>$\nu_9 = (1100)$, $\nu_{10} = (0101)$</td>
<td>${o_1 + o_2, o_2 + o_4}$</td>
</tr>
</tbody>
</table>

Repair using *two* nodes

Say $N_1$ and $N_3$

Four pieces needed to regenerate two pieces

- $(o_1 + o_2 + o_4) + (o_1) \Rightarrow o_2 + o_4$
- $(o_3) + (o_2 + o_3) \Rightarrow o_2$
- $(o_1) + (o_2) \Rightarrow o_1 + o_2$

© 2013, A. Datta & F. Oggier, NTU Singapore
## PSRC Example

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

### Repair using two nodes
- Say $N_1$ and $N_3$
- Four pieces needed to regenerate two pieces
- \((o_1 + o_2 + o_4) + (o_1) \Rightarrow o_2 + o_4\)
- \((o_3) + (o_2 + o_4) \Rightarrow o_2\)
- \((o_1) + (o_2) \Rightarrow o_1 + o_2\)

### Repair using three nodes
- Say $N_2$, $N_3$ and $N_4$
- Three pieces needed to regenerate two pieces
- \((o_2) + (o_4) \Rightarrow o_2 + o_4\)
- \((o_1 + o_2 + o_4) + (o_4) \Rightarrow o_1 + o_2\)
Recap

Data access

Replicas

fault-tolerant data access

Erasure coded data

(MSR’s Reconstruction code)

(partial) re-encode/repair
(e.g., Self-Repairing Codes)
Recap

Data insertion

Replicas

Data

fault-tolerant data access

Erasure coded data

(partial) re-encode/repair (e.g., Self-Repairing Codes)

(MSR’s Reconstruction code)

© 2013, A. Datta & F. Oggier, NTU Singapore
Replicas

Data insertion

Data access

(partial) re-encode/repair
(e.g., Self-Repairing Codes)

Erasure coded data

pipeline insertion

e.g., data for analytics

In-network coding

e.g., multimedia

fault-tolerant data access

(Transport’s Reconstruction Code)
Inserting Redundant Data
Inserting Redundant Data

- Data insertion
Inserting Redundant Data

• Data insertion
  – Replicas can be inserted in a pipelined manner
Inserting Redundant Data

- Data insertion
  - Replicas can be inserted in a pipelined manner
Inserting Redundant Data

• Data insertion
  – Replicas can be inserted in a pipelined manner

– Traditionally, erasure coded systems used a central point of processing
Inserting Redundant Data

• Data insertion
  – Replicas can be inserted in a pipelined manner

  – Traditionally, erasure coded systems used a central point of processing

Can the process of redundancy generation be distributed among the storage nodes?
In-network coding

- Ref: In-Network Redundancy Generation for Opportunistic Speedup of Backup, Future Generation Comp. Syst. 29(1), 2013
In-network coding

- Ref: In-Network Redundancy Generation for Opportunistic Speedup of Backup, Future Generation Comp. Syst. 2013

- Motivations
  - Reduce the bottleneck at a single point
    - The “source” (or first point of processing) still needs to inject “enough information” for the network to be able to carry out the rest of the redundancy generation
In-network coding

• Ref: In-Network Redundancy Generation for Opportunistic Speedup of Backup, Future Generation Comp. Syst. 2013

• Motivations
  – Reduce the bottleneck at a single point
    • The “source” (or first point of processing) still needs to inject “enough information” for the network to be able to carry out the rest of the redundancy generation
In-network coding

• Ref: In-Network Redundancy Generation for Opportunistic Speedup of Backup, Future Generation Comp. Syst. 2013

• Motivations
  – Reduce the bottleneck at a single point
    • The “source” (or first point of processing) still needs to inject “enough information” for the network to be able to carry out the rest of the redundancy generation
  – Utilize network resources opportunistically
    • Data-centers: When network/nodes are not busy doing other things
    • P2P/F2F: When nodes are online
In-network coding

• Ref: In-Network Redundancy Generation for Opportunistic Speedup of Backup, Future Generation Comp. Syst. 2013

• Motivations
  – Reduce the bottleneck at a single point
    • The “source” (or first point of processing) still needs to inject “enough information” for the network to be able to carry out the rest of the redundancy generation
  – Utilize network resources opportunistically
    • Data-centers: When network/nodes are not busy doing other things
    • P2P/F2F: When nodes are online
In-network coding
In-network coding

- Dependencies among self-repairing coded fragments can be exploited for in-network coding!
In-network coding

• Dependencies among self-repairing coded fragments can be exploited for in-network coding!
• Consider a SRC(3,7) code with the following dependencies:
In-network coding

• Dependencies among self-repairing coded fragments can be exploited for in-network coding!
• Consider a SRC(3,7) code with the following dependencies:
  – $r_1, r_2, r_3 = r_1 + r_2, r_4, r_5 = r_1 + r_4, r_6 = r_2 + r_4, r_7 = r_1 + r_6$
In-network coding

• Dependencies among self-repairing coded fragments can be exploited for in-network coding!

• Consider a SRC(3,7) code with the following dependencies:
  – \( r_1, r_2, r_3 = r_1 + r_2, r_4, r_5 = r_1 + r_4, r_6 = r_2 + r_4, r_7 = r_1 + r_6 \)
    • Note: \( r_6 = r_1 + r_7 \), etc. also … [details in Infocom 2011 paper]
In-network coding

• Dependencies among self-repairing coded fragments can be exploited for in-network coding!
• Consider a SRC(3,7) code with the following dependencies:
  – r1, r2, r3=r1+r2, r4, r5=r1+r4, r6=r2+r4, r7=r1+r6
    • Note: r6=r1+r7, etc. also … [details in Infocom 2011 paper]
• A naïve approach
In-network coding

• Dependencies among self-repairing coded fragments can be exploited for in-network coding!
• Consider a SRC(3,7) code with the following dependencies:
  - r1, r2, r3=r1+r2, r4, r5=r1+r4, r6=r2+r4, r7=r1+r6
    • Note: r6=r1+r7, etc. also … [details in Infocom 2011 paper]
• A naïve approach
In-network coding

- Dependencies among self-repairing coded fragments can be exploited for in-network coding!

- Consider a SRC(3,7) code with the following dependencies:
  - $r_1, r_2, r_3 = r_1 + r_2, r_4, r_5 = r_1 + r_4, r_6 = r_2 + r_4, r_7 = r_1 + r_6$
    - Note: $r_6 = r_1 + r_7$, etc. also … [details in Infocom 2011 paper]

- A naïve approach
In-network coding

- Dependencies among self-repairing coded fragments can be exploited for in-network coding!
- Consider a SRC(3,7) code with the following dependencies:
  - \( r_1, r_2, r_3 = r_1 + r_2, r_4, r_5 = r_1 + r_4, r_6 = r_2 + r_4, r_7 = r_1 + r_6 \)
  - Note: \( r_6 = r_1 + r_7 \), etc. also … [details in Infocom 2011 paper]
- A naïve approach
In-network coding

• Dependencies among self-repairing coded fragments can be exploited for in-network coding!

• Consider a SRC(3,7) code with the following dependencies:
  – r1, r2, r3=r1+r2, r4, r5=r1+r4, r6=r2+r4, r7=r1+r6
    • Note: r6=r1+r7, etc. also … [details in Infocom 2011 paper]

• A naïve approach

Need a good “schedule” to insert the redundancy!
- Avoid cycles/dependencies
In-network coding

• Dependencies among self-repairing coded fragments can be exploited for in-network coding!

• Consider a SRC(3,7) code with the following dependencies:
  – r1, r2, r3=r1+r2, r4, r5=r1+r4, r6=r2+r4, r7=r1+r6
    • Note: r6=r1+r7, etc. also … [details in Infocom 2011 paper]

• A naïve approach

Need a good “schedule” to insert the redundancy!
- Avoid cycles/dependencies
Subject to “unpredictable” availability of resource!!
In-network coding

• Dependencies among self-repairing coded fragments can be exploited for in-network coding!

• Consider a SRC(3,7) code with the following dependencies:
  – r1, r2, r3=r1+r2, r4, r5=r1+r4, r6=r2+r4, r7=r1+r6
    • Note: r6=r1+r7, etc. also … [details in Infocom 2011 paper]

• A naïve approach

Need a good “schedule” to insert the redundancy!
- Avoid cycles/dependencies
Subject to “unpredictable” availability of resource!!

Turns out to be $O(n!)$ w/ Oracle
In-network coding
In-network coding

• Heuristics
  – Several other policies (such as max data) were also tried

<table>
<thead>
<tr>
<th>Policy Name</th>
<th>Source Policy</th>
<th>In-Network Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>RndFlw</td>
<td>random</td>
<td>maximum flow</td>
</tr>
<tr>
<td>RndDta</td>
<td>random</td>
<td>minimum data</td>
</tr>
<tr>
<td>MinFlw</td>
<td>minimum data</td>
<td>maximum flow</td>
</tr>
<tr>
<td>MinDta</td>
<td>minimum data</td>
<td>minimum data</td>
</tr>
</tbody>
</table>

*Table 1: Different policy combinations.*
In-network coding

Figure 3: Increment of the maximum amount of stored data (throughput) for the three different availability traces.

Figure 4: Increment of the required network traffic of the in-network redundancy generation strategy for the three different availability traces.
In-network coding
In-network coding

- RndFlw was the best heuristic
  - Among those we tried
In-network coding

• RndFlw was the best heuristic
  – Among those we tried
• Provided 40% (out of a possible 57%) bandwidth savings at source for a SRC(7,3) code
In-network coding

- RndFlw was the best heuristic
  - Among those we tried
- Provided 40% (out of a possible 57%) bandwidth savings at source for a SRC(7,3) code
  - An increase in the data-insertion throughput btw. 40-60%
In-network coding

• RndFlw was the best heuristic
  – Among those we tried

• Provided 40% (out of a possible 57%) bandwidth savings at source for a SRC(7,3) code
  – An increase in the data-insertion throughput btw. 40-60%
    • No free lunch: Increase of 20-30% overall network traffic
Recap

Data insertion

- Pipelined insertion, e.g., data for analytics

Replicas

Data fault-tolerant access

- Erasure coded data
- (partial) re-encode/repair (e.g., Self-Repairing Codes)

Data access

- In-network coding, e.g., multimedia
- (MSR’s Reconstruction code)
Replicas

Data insertion

pipeline insertion
e.g., data for analytics

In-network coding
e.g., multimedia

Erasure coded data

(partial) re-encode/repair
(e.g., Self-Repairing Codes)

Replicas

archival of "cold" data

fault-tolerant data access

(MSR’s Reconstruction code)
RapidRAID

• Ref:
  – RapidRAID: Pipelined Erasure Codes for Fast Data Archival in Distributed Storage Systems (Infocom 2013)
    • Has some local repairability properties, but that aspect is yet to be explored
  – Another code instance @ ICDCN 2013
    • Decentralized Erasure Coding for Efficient Data Archival in Distributed Storage Systems
      – Systematic code (unlike RapidRAID)
      – Found using numerical methods, and a general theory for the construction of such codes, as well as their repairability properties are open issues
RapidRAID

- Ref:
  - RapidRAID: Pipelined Erasure Codes for Fast Data Archival in Distributed Storage Systems (Infocom 2013)
    - Has some local repairability properties, but that aspect is yet to be explored
  - Another code instance @ ICDCN 2013
    - Decentralized Erasure Coding for Efficient Data Archival in Distributed Storage Systems
      - Systematic code (unlike RapidRAID)
      - Found using numerical methods, and a general theory for the construction of such codes, as well as their repairability properties are open issues

- Problem statement: Can the existing (replication based) redundancy be exploited to create an erasure coded archive?
Slight change of view

Two ways to look at replicated data
RapidRAID

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.
RapidRAID

Decentralizing the hitherto centralized encoding process

Fig. 1. Network flow required to encode a data object using a classical Fig. 2. Network flow required to encode a data object using a (8,4) pipelined systematic (8,4) erasure code using nodes n1...n8. The ⊗ symbol denotes a erasure code using node n1...n8. The ⊗ symbol denotes a coding operation.
RapidRAID – Example (8,4) code
RapidRAID – Example (8,4) code

- Initial configuration

  node 1: $o_1$,  node 2: $o_2$,  node 3: $o_3$,  node 4: $o_4$,  
node 5: $o_1$,  node 6: $o_2$,  node 7: $o_3$,  node 8: $o_4$.  

RapidRAID – Example (8,4) code

• Initial configuration

  node 1: o₁,  node 2: o₂,  node 3: o₃,  node 4: o₄,
  node 5: o₁,  node 6: o₂,  node 7: o₃,  node 8: o₄.

• Logical phase 1: Pipelined coding

  \[ x_{1,2} = o₁ψ₁, \]
  \[ x_{2,3} = x_{1,2} + o₂ψ₂ = o₁ψ₁ + o₂ψ₂, \]
  \[ x_{3,4} = x_{2,3} + o₃ψ₃ = o₁ψ₁ + o₂ψ₂ + o₃ψ₃, \]
  \[ x_{4,5} = x_{3,4} + o₄ψ₄ \]
  \[ = o₁ψ₁ + o₂ψ₂ + o₃ψ₃ + o₄ψ₄, \]
  \[ x_{5,6} = x_{4,5} + o₁ψ₅ \]
  \[ = o₁(ψ₁ + ψ₅) + o₂ψ₂ + o₃ψ₃ + o₄ψ₄, \]
  \[ x_{6,7} = x_{5,6} + o₂ψ₆ \]
  \[ = o₁(ψ₁ + ψ₅) + o₂(ψ₂ + ψ₆) + o₃ψ₃ + o₄ψ₄, \]
  \[ x_{7,8} = x_{6,7} + o₃ψ₇ \]
  \[ = o₁(ψ₁ + ψ₅) + o₂(ψ₂ + ψ₆) + o₃(ψ₃ + ψ₇) + o₄ψ₄, \]
RapidRAID – Example (8,4) code
RapidRAID – Example (8,4) code

• Logical phase 2: Further local coding

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

\( \psi_i, \xi_i \) are static predetermined values
RapidRAID – Example (8,4) code

• Logical phase 2: Further local coding

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

\[ \begin{bmatrix} \xi_1 \\ \psi_1 \\ \psi_1 + \xi_5 \\ \psi_1 + \psi_5 \\ \psi_1 + \psi_5 \psi_2 + \psi_6 \\ \psi_1 + \psi_5 \psi_2 + \psi_6 \psi_3 + \psi_7 \\ \psi_1 + \psi_5 \psi_2 + \psi_6 \psi_3 + \psi_7 \psi_4 + \xi_8 \end{bmatrix} \cdot \begin{bmatrix} o_1 \\ o_2 \\ o_3 \\ o_4 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \end{bmatrix} \]

\[ \psi_i, \xi_i \text{ are static predetermined values} \]
RapidRAID: Some results

**Fig. 3.** Evaluation of the linear dependencies in \((n, k)\) RapidRAID codewords. We consider three \(n\) values with all the possible \(k\) values such that \(\frac{n}{2} \leq k < n\).

**TABLE I**

<table>
<thead>
<tr>
<th>Scheme</th>
<th>(p=0.2)</th>
<th>(p=0.1)</th>
<th>(p=0.01)</th>
<th>(p=0.001)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-replica system</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>(16,11) classical EC</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>(16,11) RapidRAID</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>11</td>
</tr>
</tbody>
</table>
RapidRAID: Some results

Fig. 5. Coding time for a (16,11) RapidRAID code. Congested nodes have 500Mbps connections with a latency of 100ms±10ms. Each candle depicts the median value, the 25-75% percentiles and the max-min values.

Fig. 6. Coding time for a (16,11) Cauchy Reed-Solomon code. Congested nodes have 500Mbps connections with a latency of 100ms±10ms. Each candle depicts the median value, the 25-75% percentiles and the max-min values.
Agenda: A composite system achieving all these properties …
Wrapping up: A moment of reflection

• Revisiting repairability – an engineering alternative
  – Redundantly Grouped Cross-object Coding for Repairable Storage (APSys2012)
    • HDFS-RAID compatible implementation
    • http://sands.sce.ntu.edu.sg/StorageCORE/
Wrapping up: A moment of reflection

Erasure coding of individual objects

RAID-4 of erasure coded pieces of different objects

(reminiscent of product codes!)
Separation of concerns

- Two distinct design objectives for distributed storage systems
  - Fault-tolerance
  - Repairability
- An extremely simple idea
  - Introduce two different kinds of redundancy
    - Any (standard) erasure code
      - for fault-tolerance
    - RAID-4 like parity (across encoded pieces of different objects)
      - for repairability
CORE repairability

• Choosing a suitable $m < k$
  – Reduction in data transfer for repair
  – Repair fan-in disentangled from base code parameter “k”
    • Large “k” may be desirable for faster (parallel) data access
    • Codes typically have trade-offs between repair fan-in, code parameter “k” and code’s storage overhead $(n/k)$

• However: The gains from reduced fan-in is probabilistic
  – For i.i.d. failures with probability “$f$”
  \[
  \Delta = m \cdot (1 - f)^m + k \cdot (1 - (1 - f)^m)
  \]

• Possible to reduce repair time
  – By pipelining data through the live nodes, and computing partial parity
Interested to

- Follow: [http://sands.sce.ntu.edu.sg/CodingForNetworkedStorage/](http://sands.sce.ntu.edu.sg/CodingForNetworkedStorage/)
  - Also, two surveys on (repairability of) storage codes
    - one detailed (FnT, June 2013)
  - Get involved: {anwitaman,frederique}@ntu.edu.sg