
Yunfeng Zhu\textsuperscript{1}, \textbf{Patrick P. C. Lee}\textsuperscript{2}, Yuchong Hu\textsuperscript{2}, Liping Xiang\textsuperscript{1}, Yinlong Xu\textsuperscript{1}

\textsuperscript{1}University of Science and Technology of China  
\textsuperscript{2}The Chinese University of Hong Kong

MSST’12
Modern Storage Systems

- Large-scale storage systems have seen deployment in practice
  - Cloud storage
  - Data centers
  - P2P storage

- Data is distributed over a collection of disks
  - Disk → physical storage device
How to Ensure Data Reliability?

- Disks can crash or have bad data
- Data reliability is achieved by keeping data redundancy across disks
  - Replication
    - Efficient computation
    - High storage overhead
  - Erasure codes (e.g., Reed-Solomon codes)
    - Less storage overhead than replication, with same fault tolerance
    - More expensive computation than replication
XOR-Based Erasure Codes

- XOR-based erasure codes
  - Encoding/decoding involve XOR operations only
  - Low computational overhead

- Different redundancy levels
  - 2-fault tolerant: RDP, EVENODD, X-Code
  - 3-fault tolerant: STAR
  - General-fault tolerant: Cauchy Reed-Solomon (CRS)
Example

- EVENODD, where number of disks = 4

\[ \begin{align*}
    a &= c + (a+c) \\
    b &= d + (b+d)
\end{align*} \]

**Note:** “+” denotes XOR operation
Failure Recovery Problem

- Recovering disk failures is necessary
  - Preserve the required redundancy level
  - Avoid data unavailability

- Single-disk failure recovery
  - Single-disk failure occurs more frequently than a concurrent multi-disk failure

- One objective of efficient single-disk failure recovery: *minimize the amount of data being read from surviving disks*
Related Work

- **Hybrid recovery**
  - Minimize amount of data being read for double-fault tolerant XOR-based erasure codes
    - e.g., RDP [Xiang, ToS’11], EVENODD [Wang, Globecom’10], X-Code [Xu, Tech Report’11]

- **Enumeration recovery** [Khan, FAST’12]
  - Enumerate all recovery possibilities to achieve optimal recovery for general XOR-based erasure codes

- **Regenerating codes** [Dimakis, ToIT’10]
  - Disks encode data during recovery
  - Minimize recovery bandwidth
Example: Recovery in RDP

- RDP with 8 disks.

<table>
<thead>
<tr>
<th>Disk0</th>
<th>Disk1</th>
<th>Disk2</th>
<th>Disk3</th>
<th>Disk4</th>
<th>Disk5</th>
</tr>
</thead>
<tbody>
<tr>
<td>d0,0</td>
<td>d0,1</td>
<td>d0,2</td>
<td>d0,3</td>
<td>d0,4</td>
<td>d0,5</td>
</tr>
<tr>
<td>d1,0</td>
<td>d1,1</td>
<td>d1,2</td>
<td>d1,3</td>
<td>d1,4</td>
<td>d1,5</td>
</tr>
<tr>
<td>d2,0</td>
<td>d2,1</td>
<td>d2,2</td>
<td>d2,3</td>
<td>d2,4</td>
<td>d2,5</td>
</tr>
<tr>
<td>d3,0</td>
<td>d3,1</td>
<td>d3,2</td>
<td>d3,3</td>
<td>d3,4</td>
<td>d3,5</td>
</tr>
<tr>
<td>d4,0</td>
<td>d4,1</td>
<td>d4,2</td>
<td>d4,3</td>
<td>d4,4</td>
<td>d4,5</td>
</tr>
<tr>
<td>d5,0</td>
<td>d5,1</td>
<td>d5,2</td>
<td>d5,3</td>
<td>d5,4</td>
<td>d5,5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Disk6</th>
<th>Disk7</th>
</tr>
</thead>
<tbody>
<tr>
<td>d0,6</td>
<td>d0,7</td>
</tr>
<tr>
<td>d1,6</td>
<td>d1,7</td>
</tr>
<tr>
<td>d2,6</td>
<td>d2,7</td>
</tr>
<tr>
<td>d3,6</td>
<td>d3,7</td>
</tr>
<tr>
<td>d4,6</td>
<td>d4,7</td>
</tr>
<tr>
<td>d5,6</td>
<td>d5,7</td>
</tr>
</tbody>
</table>

Let’s say Disk0 fails. How do we recover Disk0?
Conventional Recovery

- **Idea**: use only row parity sets. Recover each lost data symbol independently.

<table>
<thead>
<tr>
<th>Disk0</th>
<th>Disk1</th>
<th>Disk2</th>
<th>Disk3</th>
<th>Disk4</th>
<th>Disk5</th>
<th>Disk6</th>
<th>Disk7</th>
</tr>
</thead>
<tbody>
<tr>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Total number of read symbols: 36*
Hybrid Recovery

➢ Idea: use a combination of row and diagonal parity sets to maximize overlapping symbols

Total number of read symbols: 27
**Enumeration Recovery**

**Conventional Recovery**

*download 4 symbols (D2, D3, C0, C1) to recover D0 and D1*

<table>
<thead>
<tr>
<th>Recovery Equations for D0</th>
<th>Recovery Equations for D1</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0 D2 C0</td>
<td>D1 D3 C1</td>
</tr>
<tr>
<td>D0 D3 C2</td>
<td></td>
</tr>
<tr>
<td>D0 D3 C0 C1 C3</td>
<td>D1 D2 C0 C1 C2</td>
</tr>
<tr>
<td></td>
<td>D1 D2 C3</td>
</tr>
<tr>
<td>D0 D2 C1 C2 C3</td>
<td>D1 D3 C0 C2 C3</td>
</tr>
</tbody>
</table>
Challenges

- Hybrid recovery cannot be easily generalized to STAR and CRS codes, due to different data layouts
- Enumeration recovery has exponential computational overhead
- Can we develop an efficient scheme for efficient single-disk failure recovery?
Our Work

Speedup of single-disk failure recovery for XOR-based erasure codes

- Speedup in three aspects:
  - Minimize search time for returning a recovery solution
  - Minimize I/Os for recovery (hence minimize recovery time)
  - Can be extended for parallelized recovery using multi-core technologies

- Applications: when no pre-computations are available, or in online recovery
Our Work

- Design a **replace recovery** algorithm
  - Hill-climbing approach: incrementally replace feasible recovery solutions with fewer disk reads

- Implement and experiment on a networked storage testbed
  - Show recovery time reduction in both single-threaded and parallelized implementation
Key Observation

There likely exists an optimal recovery solution, such that this solution has exactly $\omega$ parity symbols!

A strip of $\omega$ data symbols is lost
To recover a failed disk, choose a collection of parity symbols (per stripe) such that:

- The collection has $\omega$ parity symbols
- The collection can correctly resolve the $\omega$ lost data symbols
- Total number of data symbols encoded in the $\omega$ parity symbols is minimum $\rightarrow$ minimize disk reads
Replace Recovery Algorithm

Notation:

<table>
<thead>
<tr>
<th>$P_i$</th>
<th>set of parity symbols in the $i$th ($1 \leq i \leq m$) parity disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>collection of $\omega$ parity symbols used for recovery</td>
</tr>
<tr>
<td>$Y$</td>
<td>collection of parity symbols that are considered to be included in $X$</td>
</tr>
</tbody>
</table>

Algorithm:

1. Initialize $X$ with the $\omega$ parity symbols of $P_1$
2. Set $Y$ to be the collection of parity symbols in $P_2$; Replace “some” parity symbols in $X$ with same number of symbols in $Y$, such that $X$ is valid to resolve the $\omega$ lost data symbols
3. Replace Step 2 by resetting $Y$ with $P_3, \ldots, P_m$
4. Obtain resulting $X$ and corresponding encoding data symbols

Target: reduce number of read symbols
**Example**

**Step 1:** Initialize $X = \{C_0, C_1\}$. Number of read symbols of $X$ is 4

**Step 2:** Consider $Y = \{C_2, C_3\}$. $C_2$ can replace $C_0$ ($X$ is valid). Number of read symbols equal to 3

**Step 3:** Replace $C_0$ with $C_2$. $X = \{C_2, C_1\}$. Note it is an optimal solution.
Algorithmic Extensions

- Replace recovery has polynomial complexity
- **Extensions**: increase search space, while maintaining polynomial complexity
  - Multiple rounds
    - Use different parity disks for initialization
  - Successive searches
    - After considering $P_i$, reconsider the previously considered $i-2$ parity symbol collections (*univariate search*)

- Can be extended for general I/O recovery cost
- Details in the paper
Evaluation: Recovery Performance

- Recovery performance for STAR

Replace recovery is close to lower bound
Evaluation: Recovery Performance

Recovery performance for CRS

\[ m = 3, \quad \omega = 4 \]

\[ m = 3, \quad \omega = 5 \]

Replace recovery is close to optimal (< 3.5% difference)
Evaluation: Search Performance

- Enumeration recovery has a huge search space
  - Maximum number of recovery equations being enumerated is $2^{m\omega}$.

- Search performance for CRS
  - Intel 3.2GHz CPU, 2GB RAM

<table>
<thead>
<tr>
<th>$(k, m, \omega)$</th>
<th>Time (Enumeration)</th>
<th>Time (Replace)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10, 3, 5)</td>
<td>6m32s</td>
<td>0.08s</td>
</tr>
<tr>
<td>(12, 4, 4)</td>
<td>17m17s</td>
<td>0.09s</td>
</tr>
<tr>
<td>(10, 3, 6)</td>
<td>18h15m17s</td>
<td>0.24s</td>
</tr>
<tr>
<td>(12, 4, 5)</td>
<td>13d18h6m43s</td>
<td>0.30s</td>
</tr>
</tbody>
</table>

Replace recovery uses significantly less search time than enumeration recovery
Design and Implementation

- Recovery thread
  - Reading data from surviving disks
  - Reconstructing lost data of failed disk
  - Writing reconstructed data to a new disk

- Parallel recovery architecture
  - Stripe-oriented recovery: each recovery thread recovers data of a stripe
  - Multi-thread, multi-server
  - Details in the paper
Experiments

- Experiments on a networked storage testbed
  - Conventional vs. Recovery
  - Default chunk size = 512KB
  - Communication via ATA over Ethernet (AoE)

- Types of disks (physical storage devices)
  - Pentium 4 PCs
  - Network attached storage (NAS) drives
  - Intel Quad-core servers
Recovery Time Performance

- Conventional vs Replace: double-fault tolerant codes:

**RDP**

- **X-Code**

**EVENODD**

- **CRS(k, m=2)**
Recovery Time Performance

- Conventional vs Replace: Triple and general-fault tolerant codes

**Star CRS(k, m=3)**

**CRS(k, m>3)**

- Graphs showing recovery time performance for different values of k and p or m.
Summary of Results

- Replace recovery reduces recovery time of conventional recovery by 10-30%.

- Impact of chunk size:
  - Larger chunk size, recovery time decreases.
  - Replace recovery still shows the recovery time reduction.

- Parallel recovery:
  - Overall recovery time reduces with multi-thread, multi-server implementation.
  - Replace recovery still shows the recovery time reduction.

- Details in the paper.
Conclusions

- Propose a replace recovery algorithm
  - provides near-optimal recovery performance for STAR and CRS codes
  - has a polynomial computational complexity

- Implement replace recovery on a parallelized architecture

- Show via testbed experiments that replace recovery speeds up recovery over conventional

- Source code:
Backup
Impact of Chunk Size

Conventional recovery

- Recovery time decreases as chunk size increases
- Recovery time stabilizes for large chunk size

Replace recovery
Parallel Recovery

➢ Recovery performance of multi-threaded implementation:
  • Recovery time decreases as number of threads increases
  • Improvement bounded by number of CPU cores
  • We show applicability of replace recovery in parallelized implementation

➢ Similar results observed in our multi-server recovery implementation

STAR (p = 13)
Quad-core case