On the Speedup of Single-Disk Failure Recovery in XOR-Coded Storage Systems: Theory and Practice

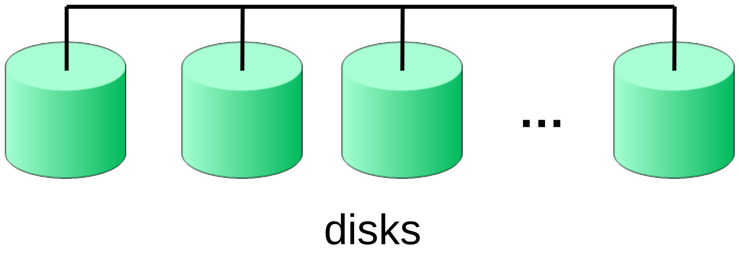
Yunfeng Zhu[1], **Patrick P. C. Lee**[2], Yuchong Hu[2], Liping Xiang[1], Yinlong Xu[1]

[1]University of Science and Technology of China
[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?

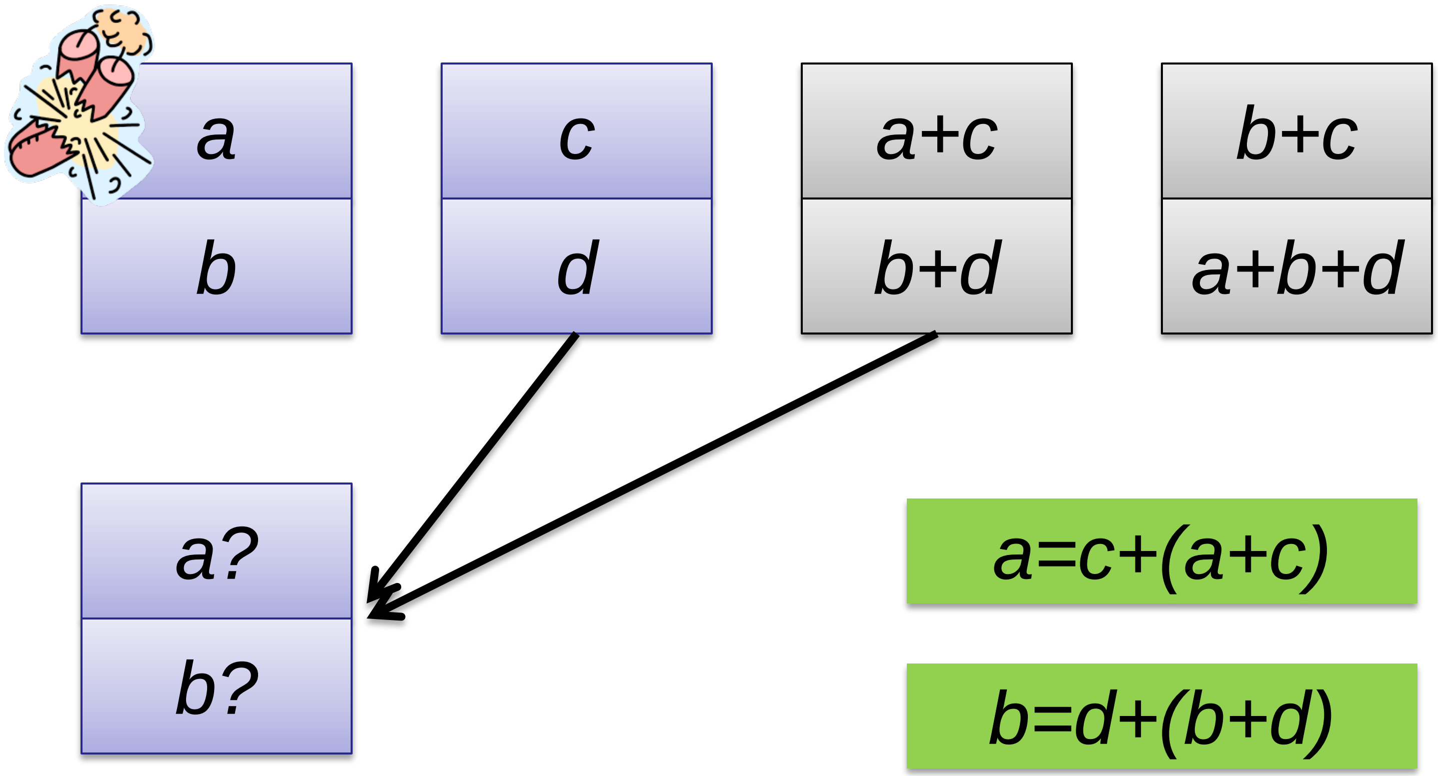
- Disks can crash or have bad data
- Data reliability is achieved by keeping **data redundancy** across disks
  - Replication
    - Efficient computation
    - High storage overhead
  - Erasure codes (e.g., Reed-Solomon codes)
    - Less storage overhead than replication, with same fault tolerance
 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



***Note**: "+" denotes XOR operation*

Failure Recovery Problem

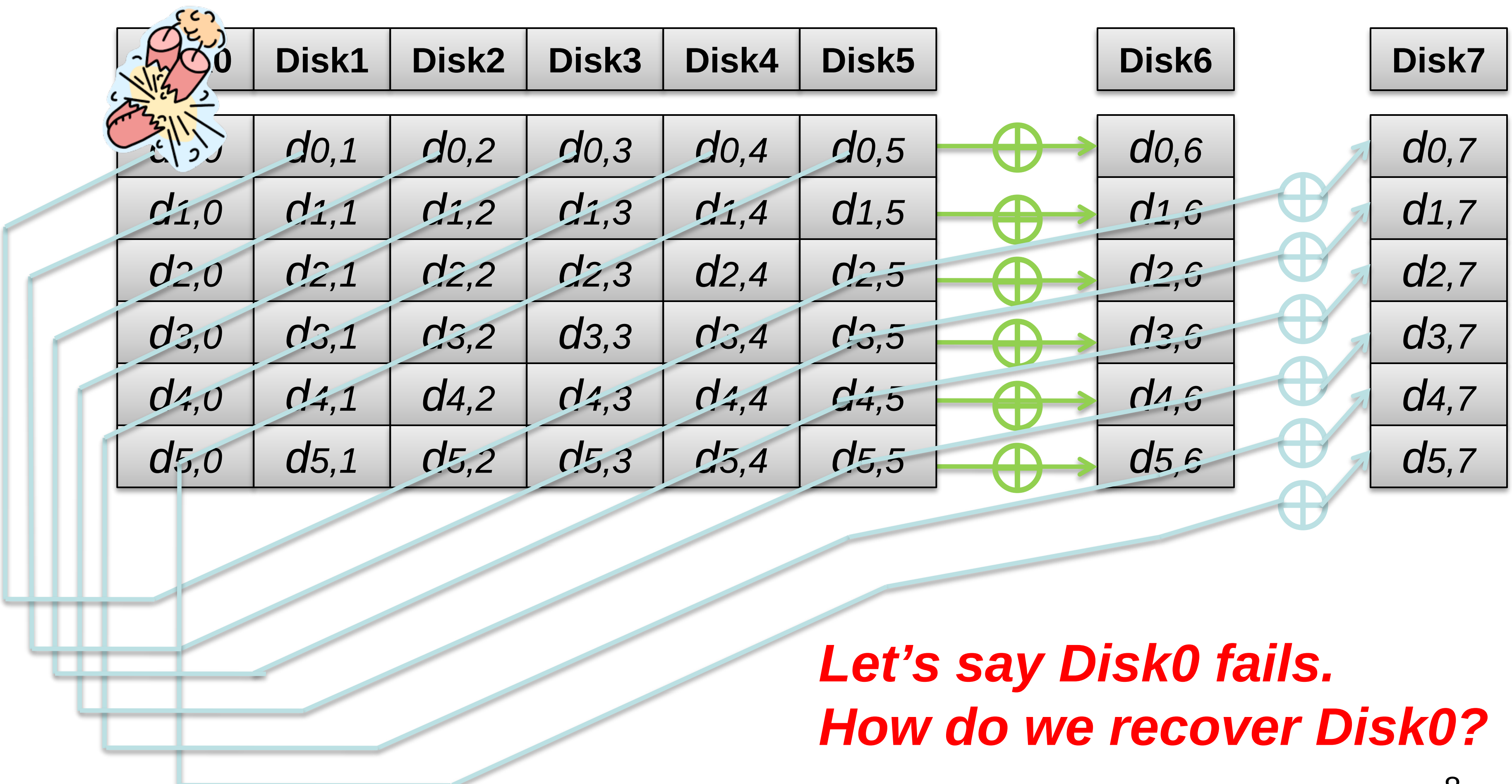
- Recovering disk failures is necessary
  - Preserve the required redundancy level
  - Avoid data unavailability
- Single-disk failure recovery
  - Single-disk failure occurs more frequently than a concurrent multi-disk failure
- One objective of efficient single-disk failure recovery: ***minimize the 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.



Let's say Disk0 fails.
How do we recover Disk0?

Conventional Recovery

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

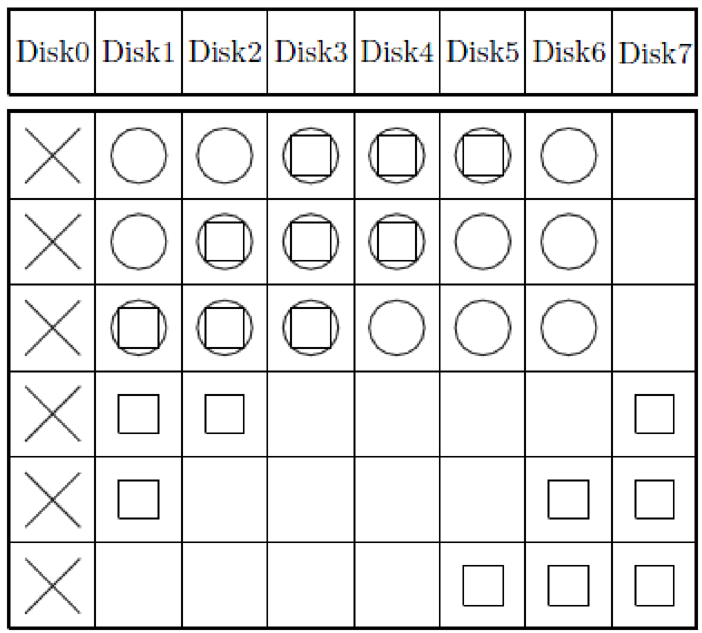
Disk0	Disk1	Disk2	Disk3	Disk4	Disk5	Disk6	Disk7
X	O	O	O	O	O	O	
X	O	O	O	O	O	O	
X	O	O	O	O	O	O	
X	O	O	O	O	O	O	
X	O	O	O	O	O	O	
X	O	O	O	O	O	O	

Total number of read symbols: ***36***

[Xiang, ToS'11]

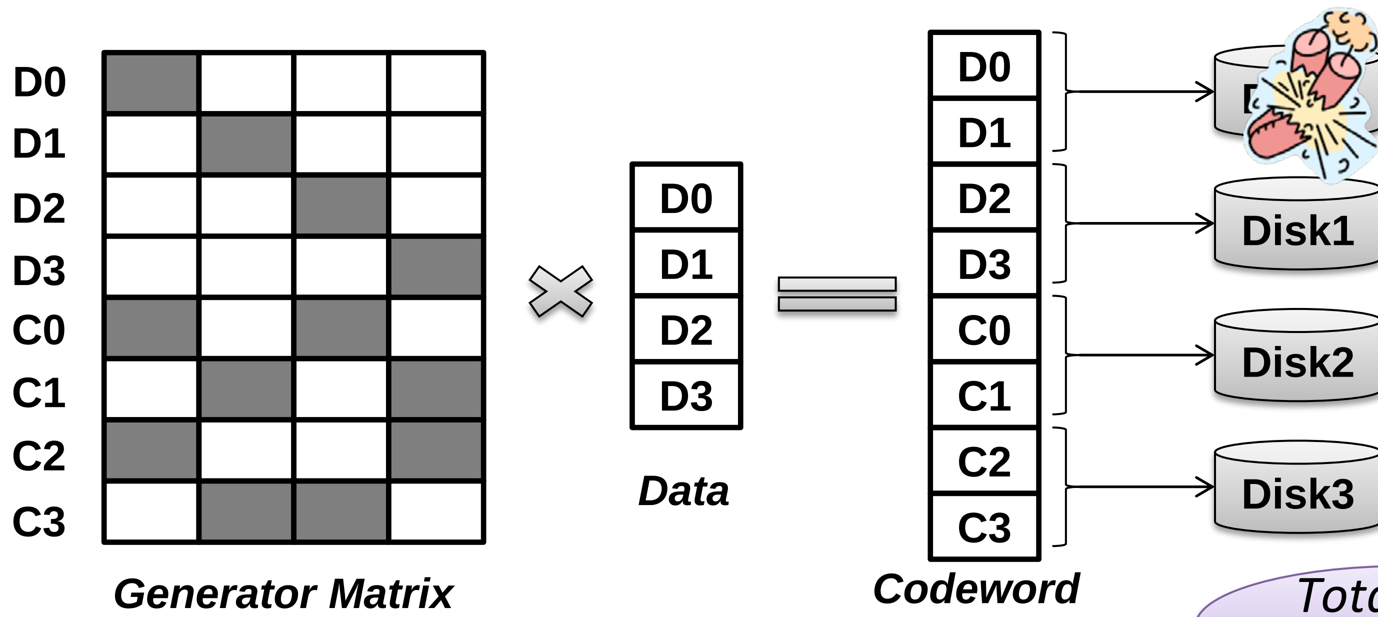
Hybrid Recovery

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



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

Enumeration Recovery



*Total read symbols: **3***

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

Recovery Equations for D0	*Recovery Equations for D1*
D0 D2 C0	D1 D3 C1
D0 D3 C2	D1 D2 C0 C1 C2
D0 D3 C0 C1 C3	D1 D2 C3
D0 D2 C1 C2 C3	D1 D3 C0 C2 C3

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

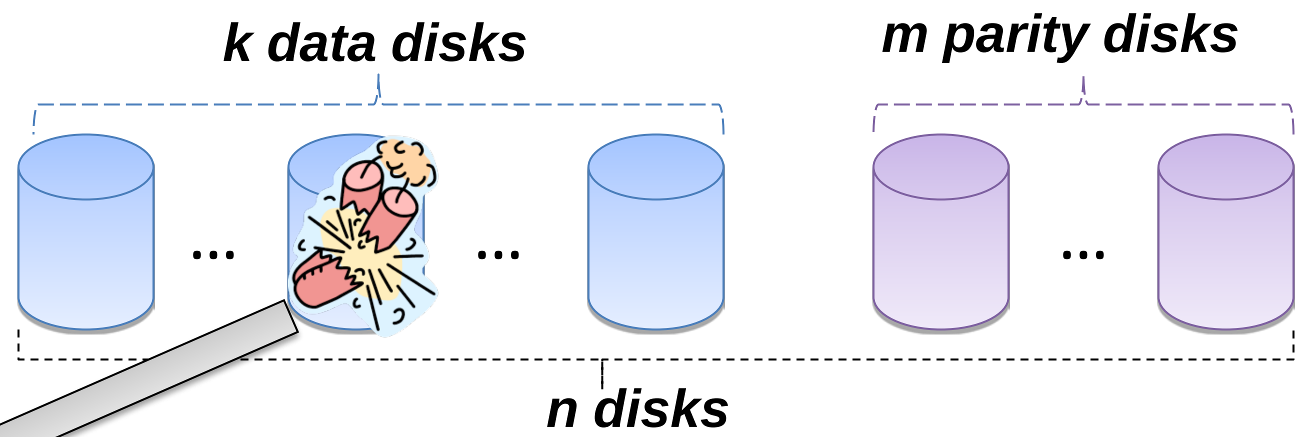
- Speedup in three aspects:
  - Minimize search time for returning a recovery solution
  - Minimize I/Os for recovery (hence minimize recovery time)
  - Can be extended for parallelized recovery using multi-core technologies
- Applications: when no pre-computations are available, or in online recovery

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

Strip size: ω



A strip of ω data symbols is lost

*There likely exists an optimal recovery solution, such that this solution has **exactly ω parity symbols**!*

Simplified Recovery Model

- To recover a failed disk, choose a collection of parity symbols (per stripe) such that:
  - The collection has ω parity symbols
  - The collection can correctly resolve the ω lost data symbols
  - Total number of data symbols encoded in the ω parity symbols is minimum → minimize disk reads

Replace Recovery Algorithm

Notation:

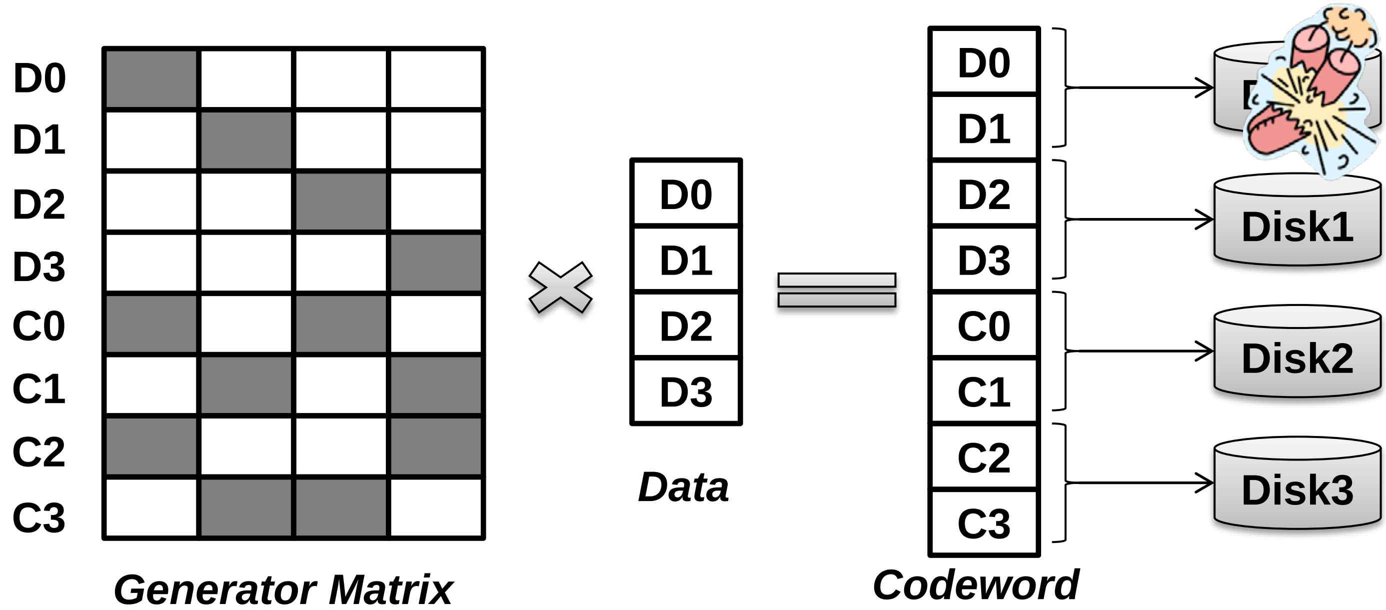
P_i	set of parity symbols in the *i*th ($1 \le i \le m$) parity disk
X	collection of ω parity symbols used for recovery
Y	collection of parity symbols that are considered to be included in ***X***

Target: *reduce number of read symbols*

Algorithm:

1	Initialize ***X*** with the ω 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 ω 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

Example



Step 1: *Initialize **X** = {**C0**, **C1**}. Number of read symbols of **X** is **4***

Step 2: *Consider **Y** = {**C2**, **C3**}. **C2** can replace **C0** (X is valid).*
*Number of read symbols equal to **3***

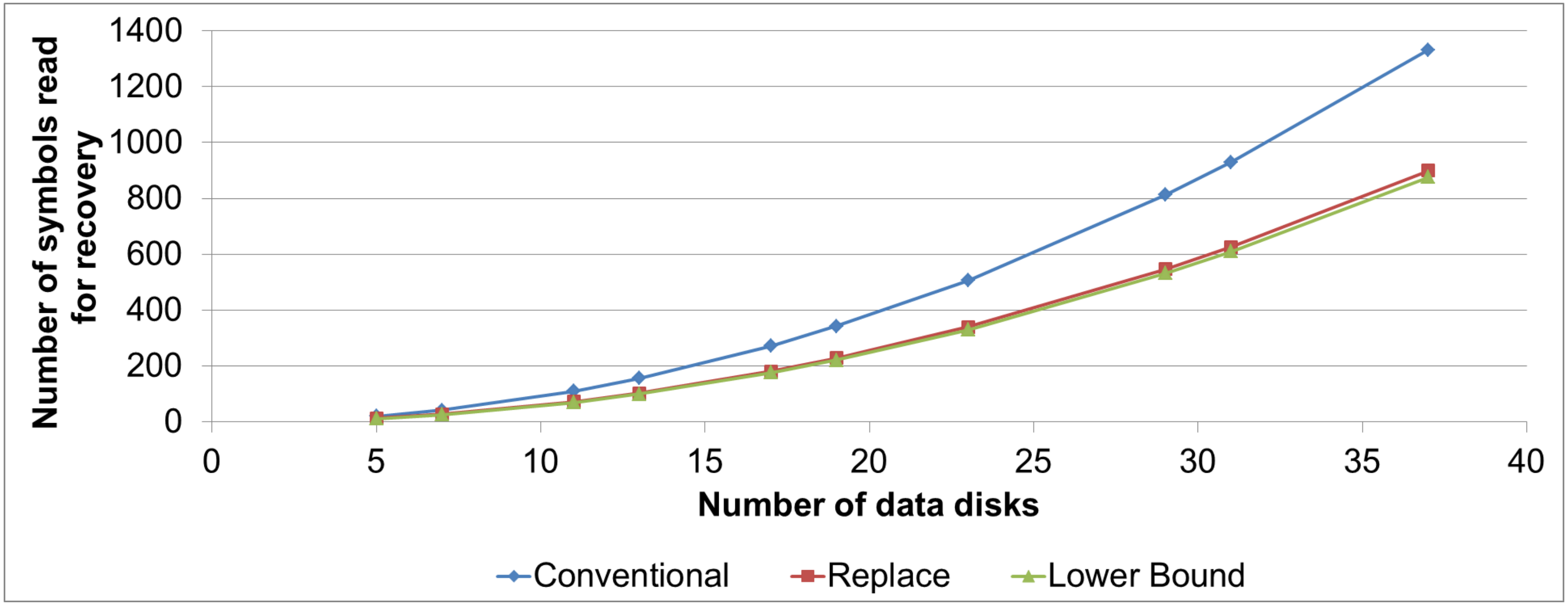
Step 3: *Replace **C0** with **C2**. **X** = {**C2**,**C1**}. Note it is an optimal solution.*

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 $\boldsymbol{P_i}$, 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

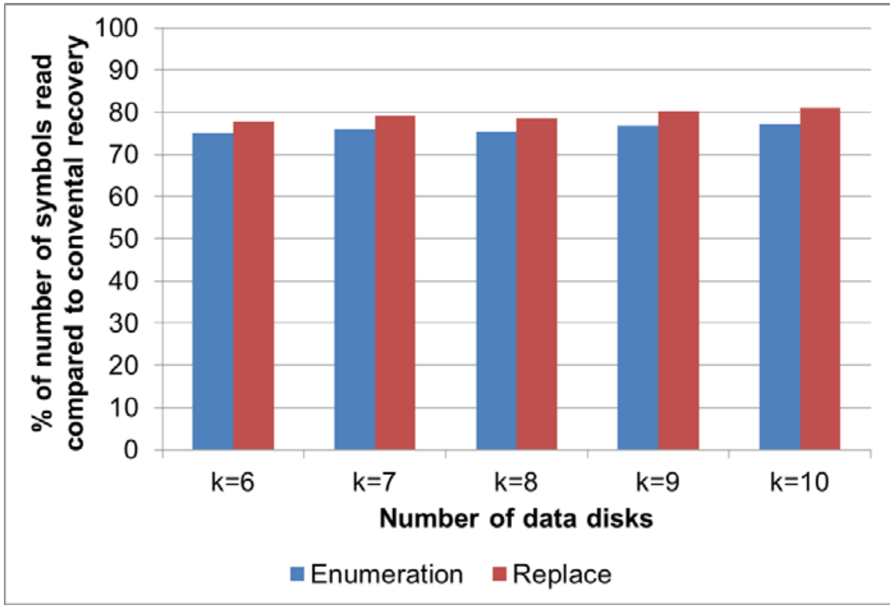


Replace recovery is close to lower bound

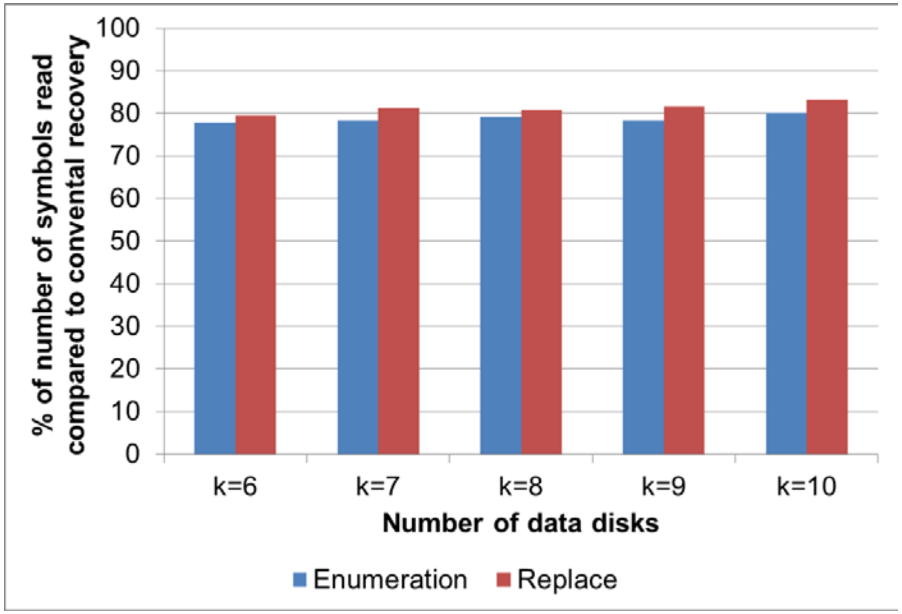
Evaluation: Recovery Performance

- Recovery performance for CRS

***m* = 3, *ω* = 4**



***m* = 3, *ω* = 5**



Replace recovery is close to optimal (< 3.5% difference)

Evaluation: Search Performance

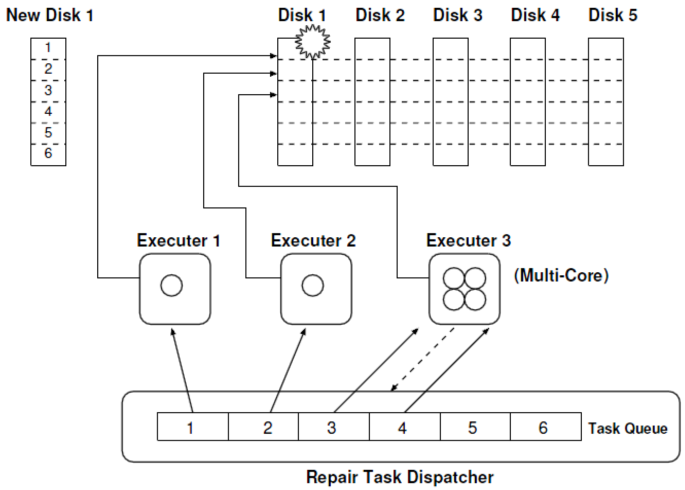
- Enumeration recovery has a huge search space
  - Maximum number of recovery equations being enumerated is $\mathbf{2}^{m\omega}$.
- Search performance for CRS
  - Intel 3.2GHz CPU, 2GB RAM

(k, m, *ω*)	Time (Enumeration)	Time (Replace)
(10, 3, 5)	6m32s	0.08s
(12, 4, 4)	17m17s	0.09s
(10, 3, 6)	18h15m17s	0.24s
(12, 4, 5)	13d18h6m43s	0.30s

Replace recovery uses significantly less search time than enumeration recovery

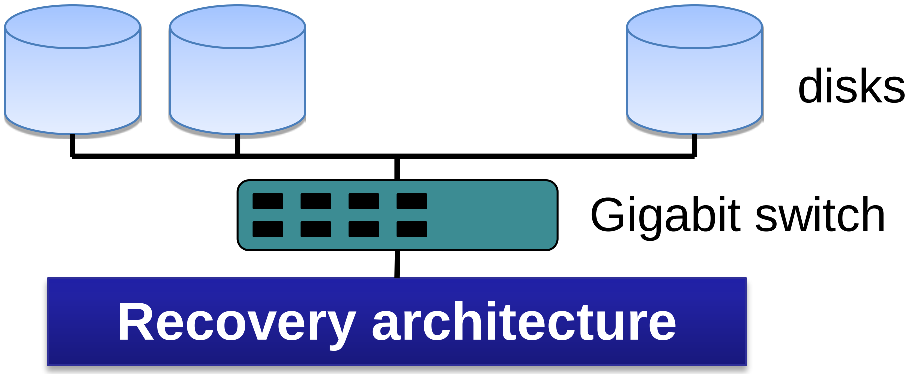
Design and Implementation

- 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)

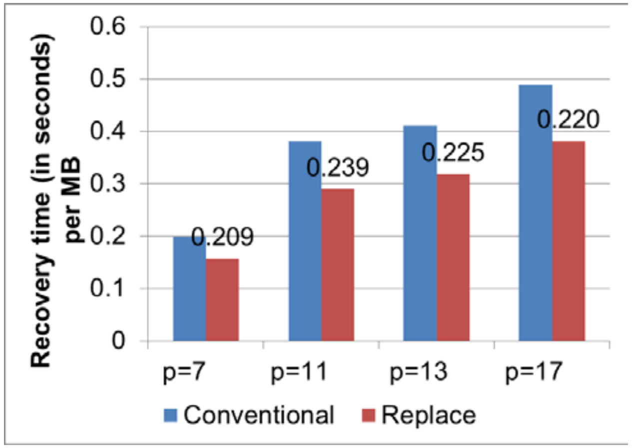


- Types of disks (physical storage devices)
  - Pentium 4 PCs
  - Network attached storage (NAS) drives
  - Intel Quad-core servers

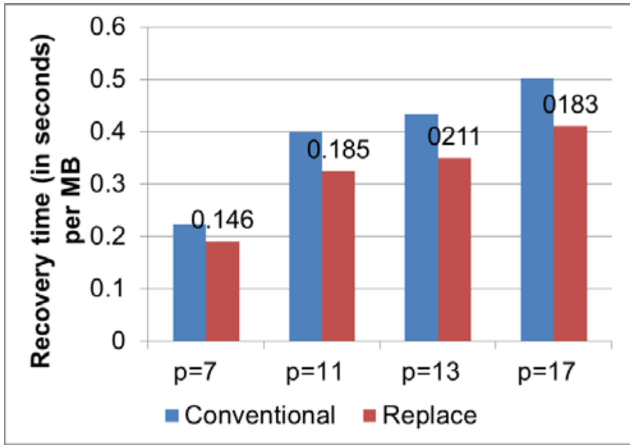
Recovery Time Performance

- Conventional vs Replace: double-fault tolerant codes:

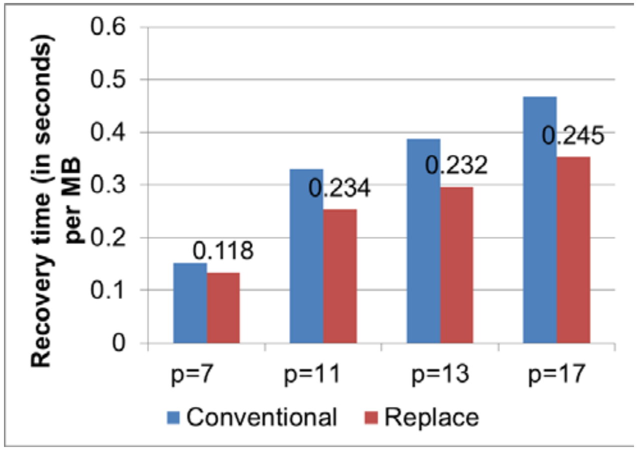
RDP



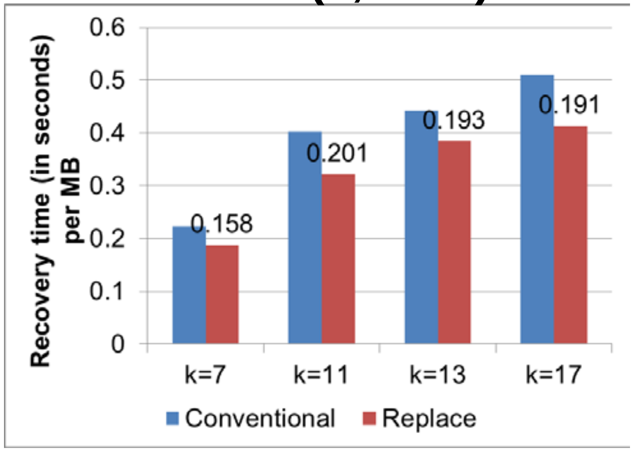
EVENODD



X-Code



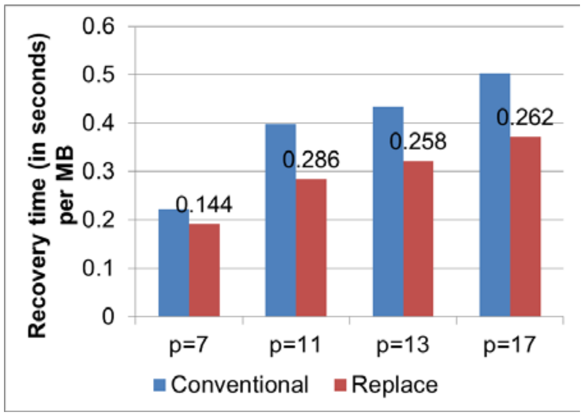
CRS(k, m=2)



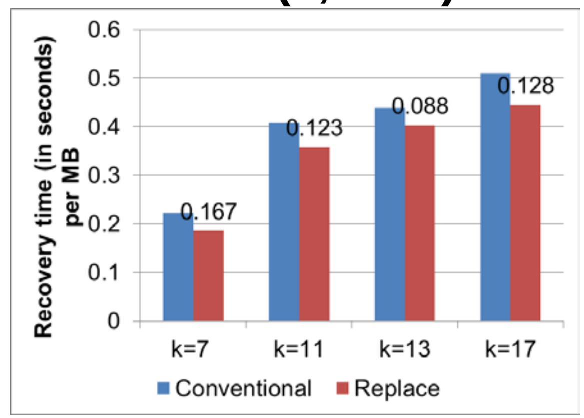
Recovery Time Performance

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

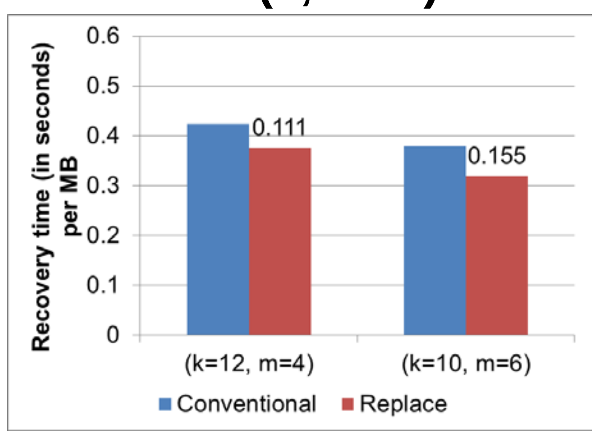
STAR



CRS(k, m=3)



CRS(k, m>3)



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

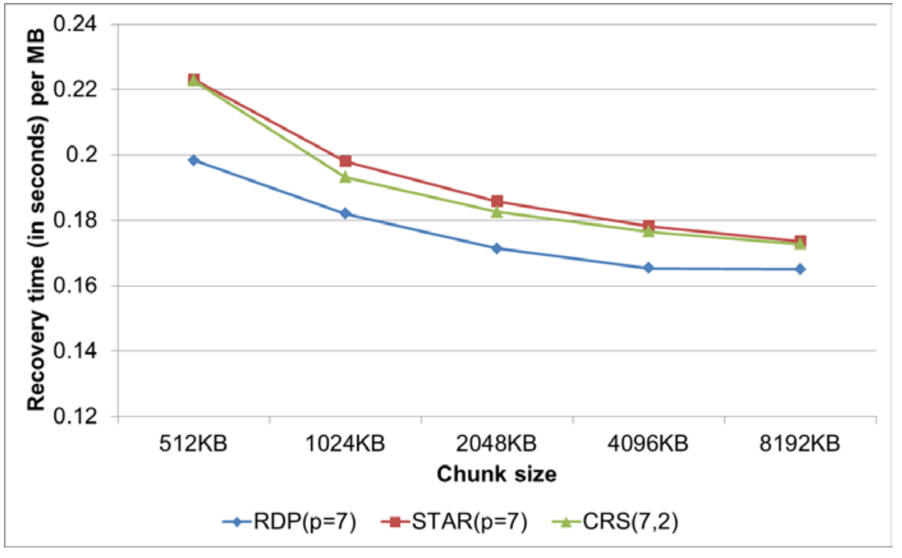
Conclusions

- 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:
  - **http://ansrlab.cse.cuhk.edu.hk/software/zpacr/**

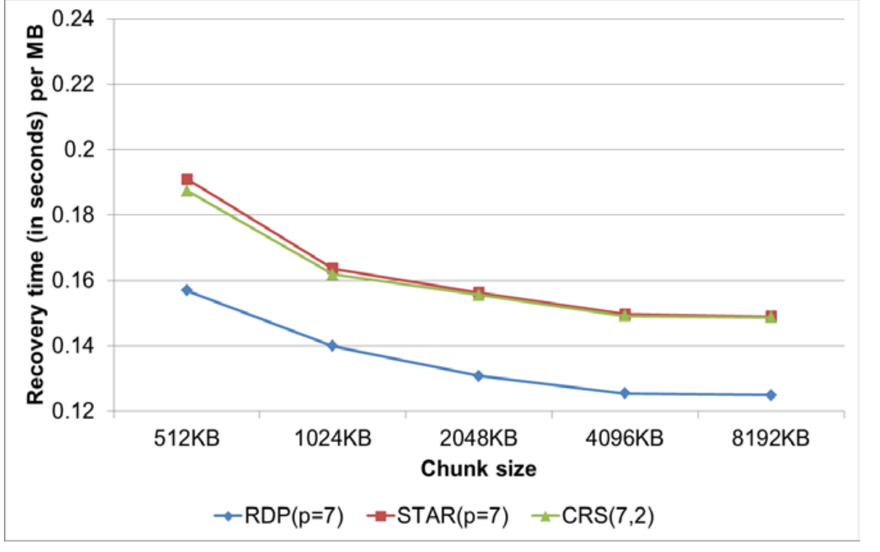
Backup

Impact of Chunk Size

Conventional recovery

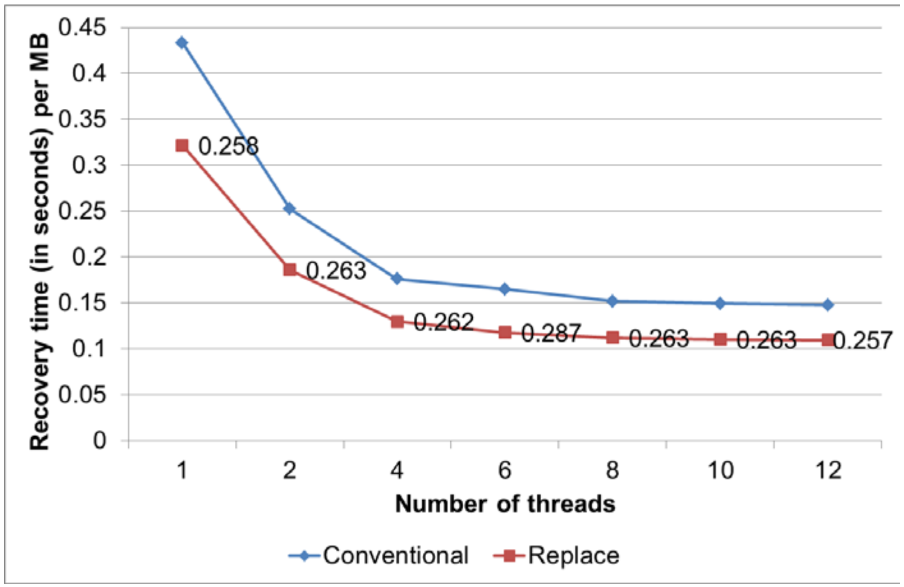


Replace recovery



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

Parallel Recovery



STAR (p = 13)
Quad-core case

- 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