

NCFS: On the Practicality and Extensibility of a Network-Coding-Based Distributed File System

Yuchong Hu¹, Chiu-Man Yu², Yan-Kit Li²
Patrick P. C. Lee², John C. S. Lui²

¹Institute of Network Coding

²Department of Computer Science and Engineering

The Chinese University of Hong Kong

NetCod '11

NCFS Overview

➤ **NCFS: Network-coding-based distributed file system**

- Use network coding to speed up repair while preserving fault tolerance of storage
- No intelligence required on storage nodes
- Data is transparently striped across nodes

NCFS Overview

- NCFS is a **file system**
 - Organize storage in hierarchical namespace
 - Manage file namespace metadata (e.g., filenames, directories, access rights)
 - Clients only see a mount drive
 - Similar to Linux/Windows file systems

```
# Login as root (now ncfs needs root access)
sudo -i

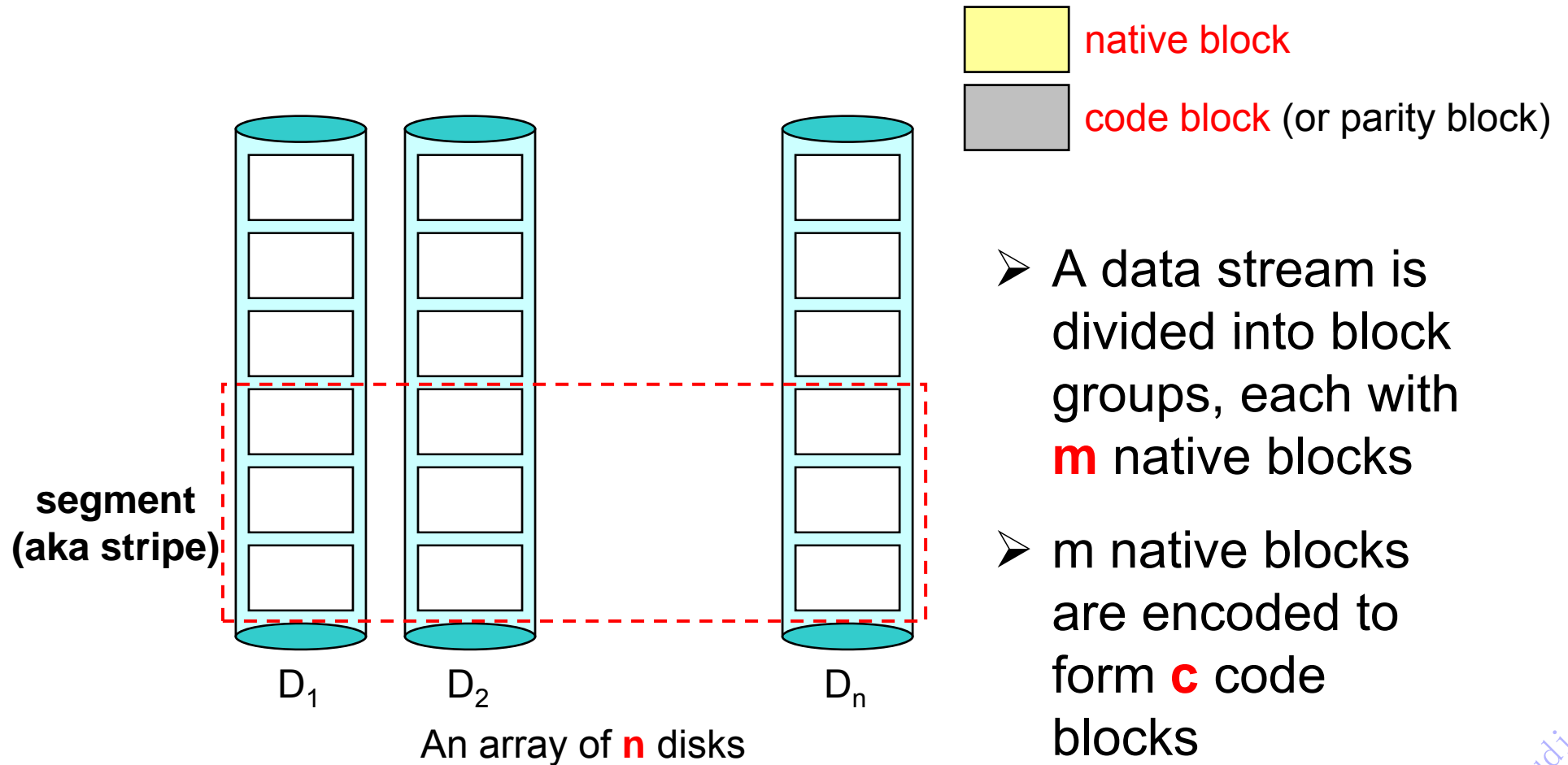
# mount file system on ./mountdir
./ncfs rootdir mountdir

# copy file to ncfs
time cp test.dat ./mountdir
```

Why We Build NCFS?

- Develop a **proof-of-concept prototype** for experimenting erasure codes and regenerating codes in practice
 - We by no means claim it's ready for large-scale use
- Provide a **storage interface** between user applications and distributed storage backends
- Provide an **extensible platform** for future systems research

Definitions



MDS Codes

- A **maximum distance separable** code, denoted by **MDS(n, k)**, has the property that any **k** ($< n$) out of n nodes can be used to reconstruct original native blocks
 - i.e., at most $n-k$ disk failures can be tolerated
- Example:
 - RAID-5 is an MDS($n, n-1$) code as it can tolerate 1 disk failure

Repair Degree

- Repair degree **d**
 - Consider the case where one disk is failed
 - The repair for the lost blocks of the failed disk is achieved by retrieving data from d disks

- The design of an MDS code is determined by the parameters **(n, k, d, m, c)**

Classes of MDS Codes

➤ Erasure codes

- Stripe data with redundancy
- e.g., RAID-like codes, array codes

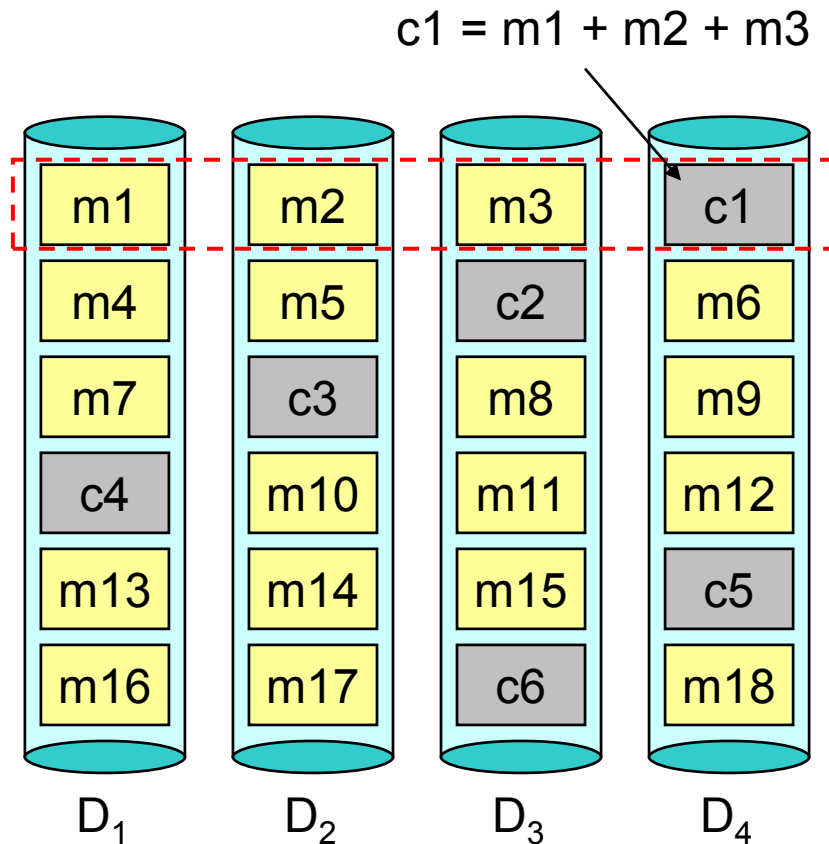
We focus on the optimal erasure codes (i.e., the MDS property is achieved). Near-optimal erasure codes require slightly more than k blocks to recover the original data

➤ Regenerating codes

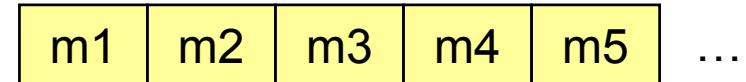
- Stripe data with redundancy *and* reduce the repair bandwidth based on network coding

Erasure Codes

➤ RAID-5



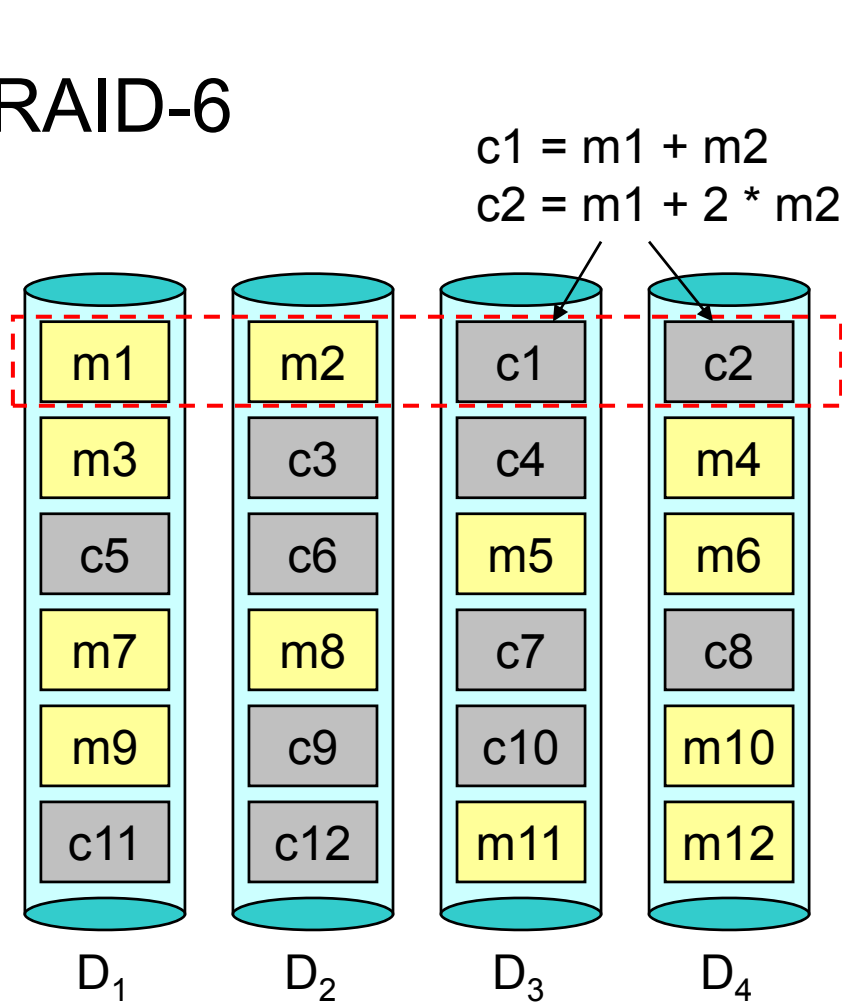
Data stream of native blocks



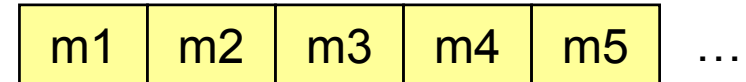
- Define $+$ as XOR operator
- Code block
 - $c1 = \text{XOR-sum of native blocks in same segment}$
- Parameters:
 - $n = \text{can be any } \geq 2$
 - $k = n - 1$
 - $d = k$
 - $m = n - 1$
 - $c = 1$

Erasure Codes

➤ RAID-6



Data stream of native blocks



➤ Code blocks:

- c1 = XOR-sum of native blocks in same segment
- c2 = Linear combination based on Reed Solomon codes

➤ Parameters:

- n = can be any ≥ 3
- k = n - 2
- d = k
- m = n - 2
- c = 2

Regenerating Codes

- **Regenerating codes** are the MDS codes that realize the optimal tradeoff between storage and repair bandwidth
- Two extreme points of the tradeoff curve:
 - MBR (minimum bandwidth regenerating codes)
 - MSR (minimum storage regenerating codes)

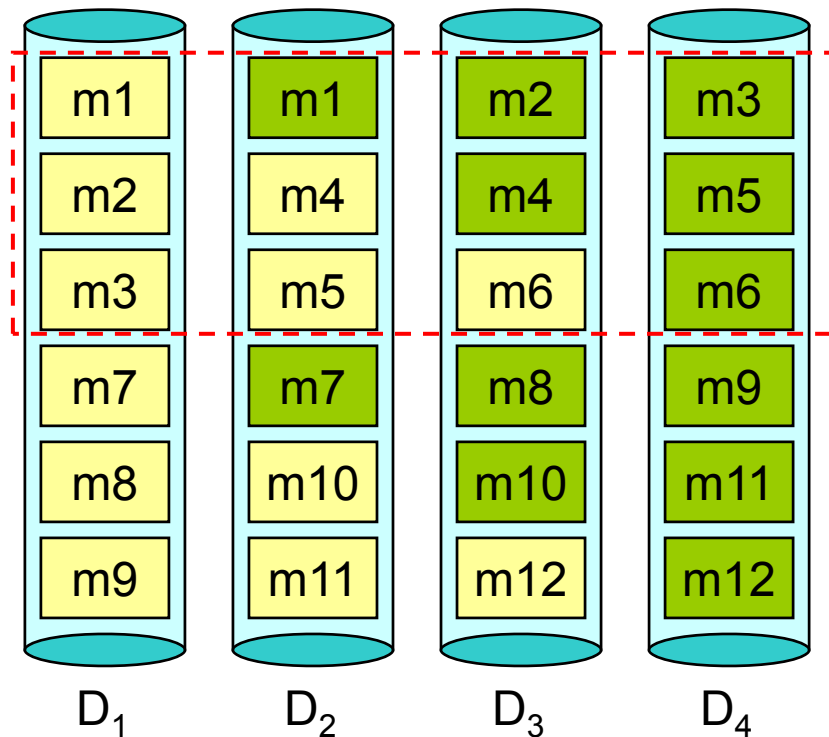
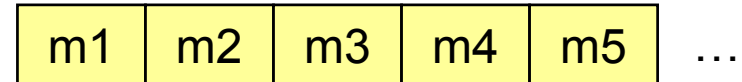
Regenerating Codes

- E-MBR (Minimum bandwidth regenerating) codes with exact repair
 - Minimize repair bandwidth while maintaining MDS property
- Idea:
 - Assume $d = n - 1$ (repair data from $n - 1$ survival disks during a single-node failure)
 - Make a duplicate copy for each native/code block
 - To repair a failed disk, download **exactly one block per segment** from each survival disk

Regenerating Codes

➤ E-MBR($n, k=n-1, d=n-1$)

Data stream of native blocks



➤ Duplicate each native block. Both native and duplicate blocks are in different disks

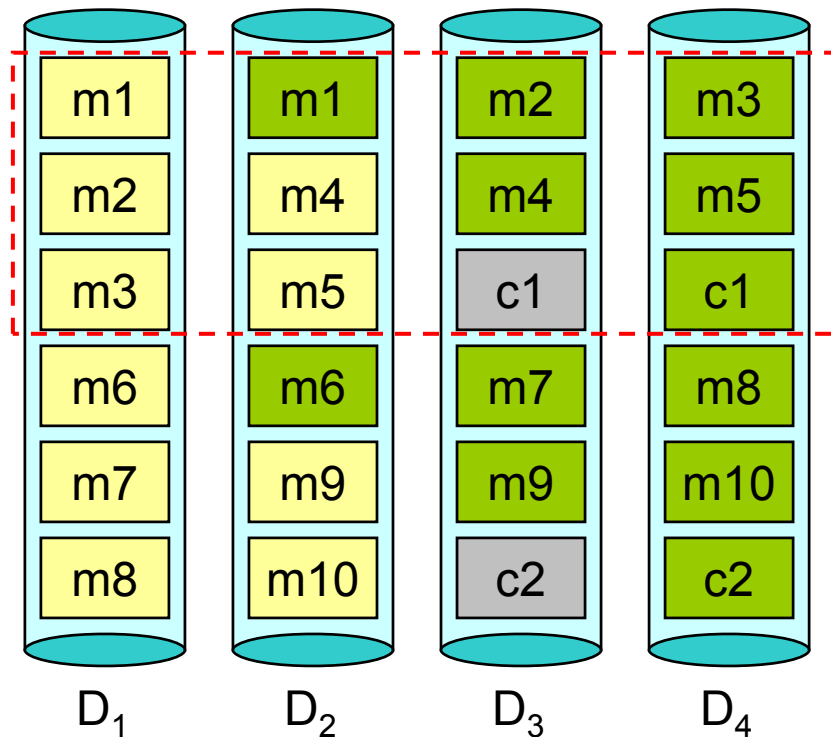
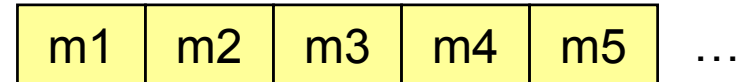
➤ Parameters:

- $n =$ can be any ≥ 2
- $k = n - 1$
- $d = n - 1$
- $m = n(n-1)/2$
- $c = 0$ (i.e., no code block)

Regenerating Codes

➤ E-MBR($n, k=n-2, d=n-1$)

Data stream of native blocks



➤ Code block:

- $c_1 = m_1 + m_2 + m_3 + m_4 + m_5$

➤ Parameters:

- $n =$ can be any ≥ 3
- $k = n - 2$
- $d = n - 1$
- $m = (n-2)(n+1)/2$
- $c = 1$

Regenerating Codes

➤ E-MBR($n, k, d=n-1$)

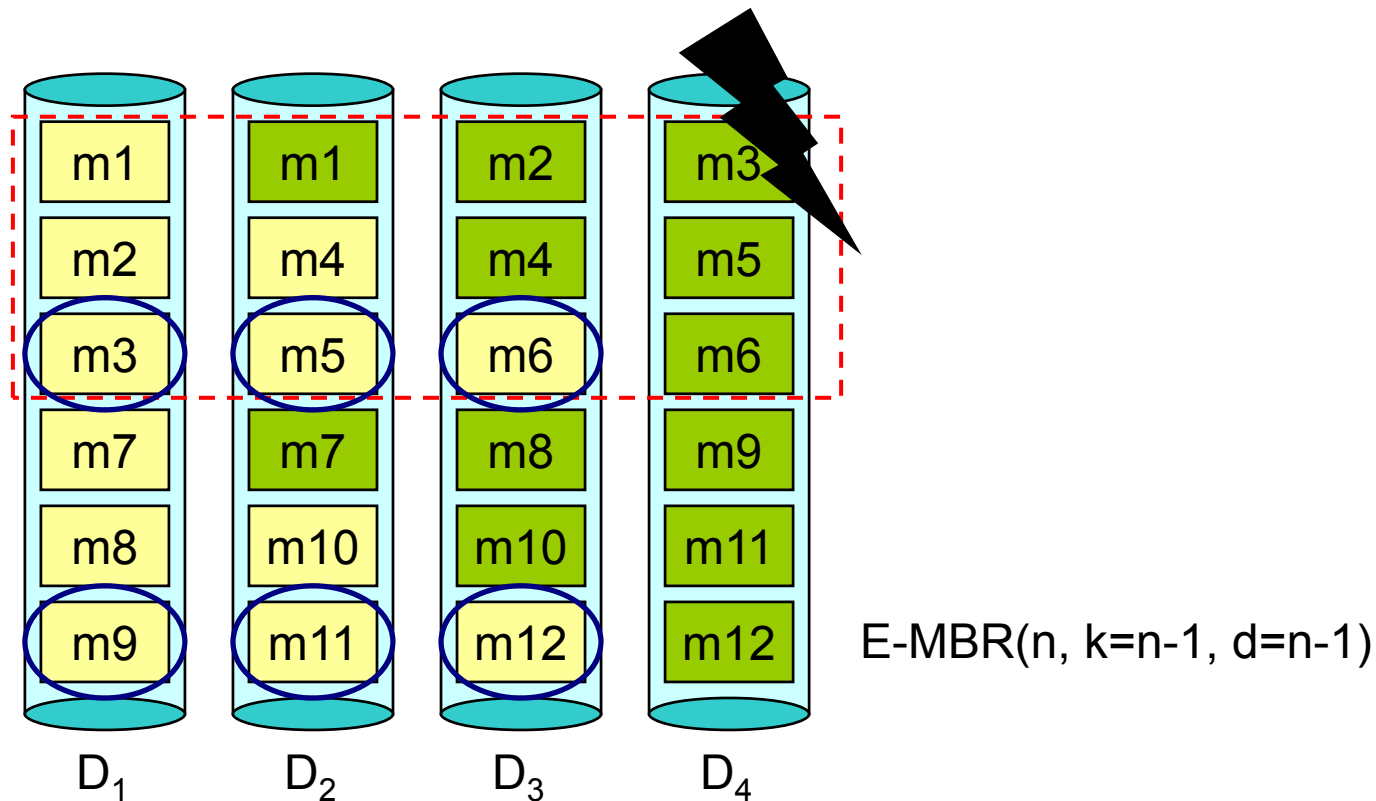
- For general n, k
- Each native/code block still has a duplicate copy
- Code blocks are formed by Reed-Solomon codes

➤ Parameters:

- $n =$ can be any $\geq k+1$
- $k =$ can be any
- $d = n - 1$
- $m = k(2n-k-1)/2$
- $c = (n - k)(n - k - 1)/2$

Regenerating Codes

- Repair in E-MBR: download one block per segment from each survival disk



Regenerating Codes

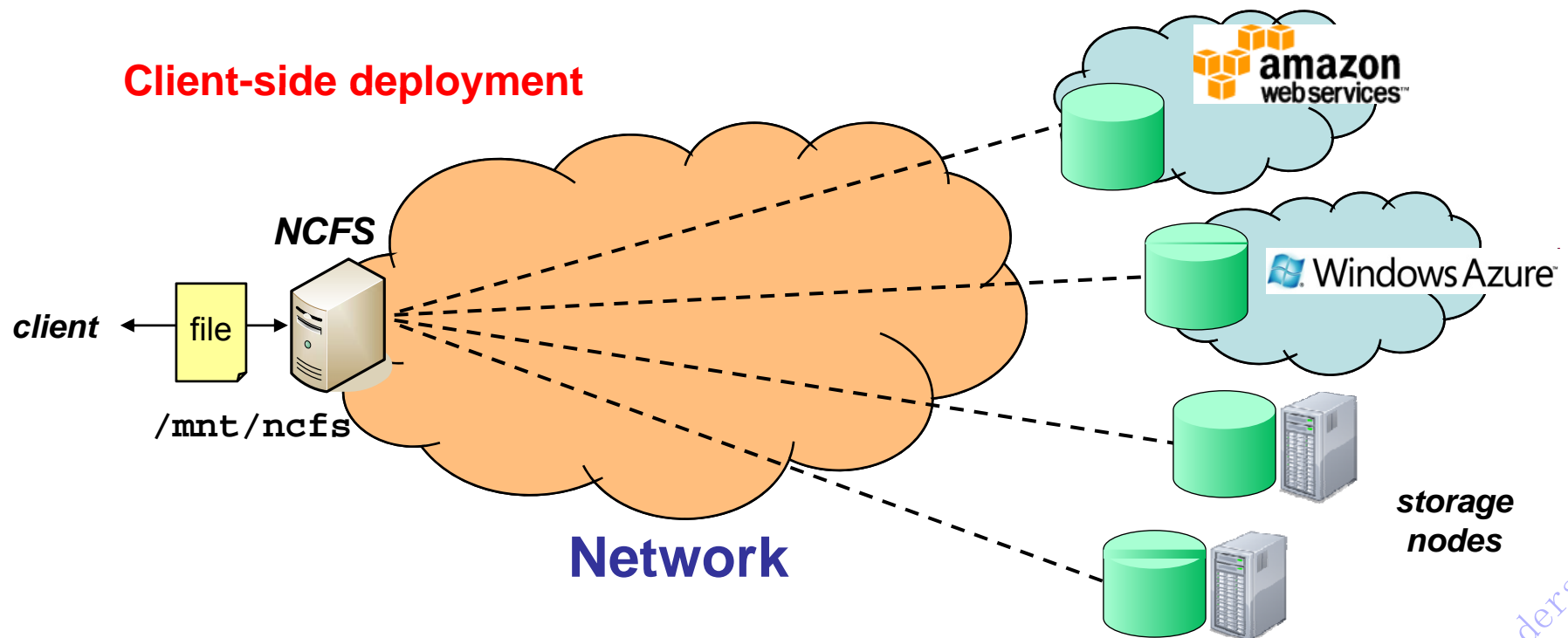
- Others regenerating codes (e.g., MSR) typically require nodes to perform encoding
- This assumption may be too strong for some types of storage nodes
 - e.g., block devices such as harddisks

Codes That We Consider

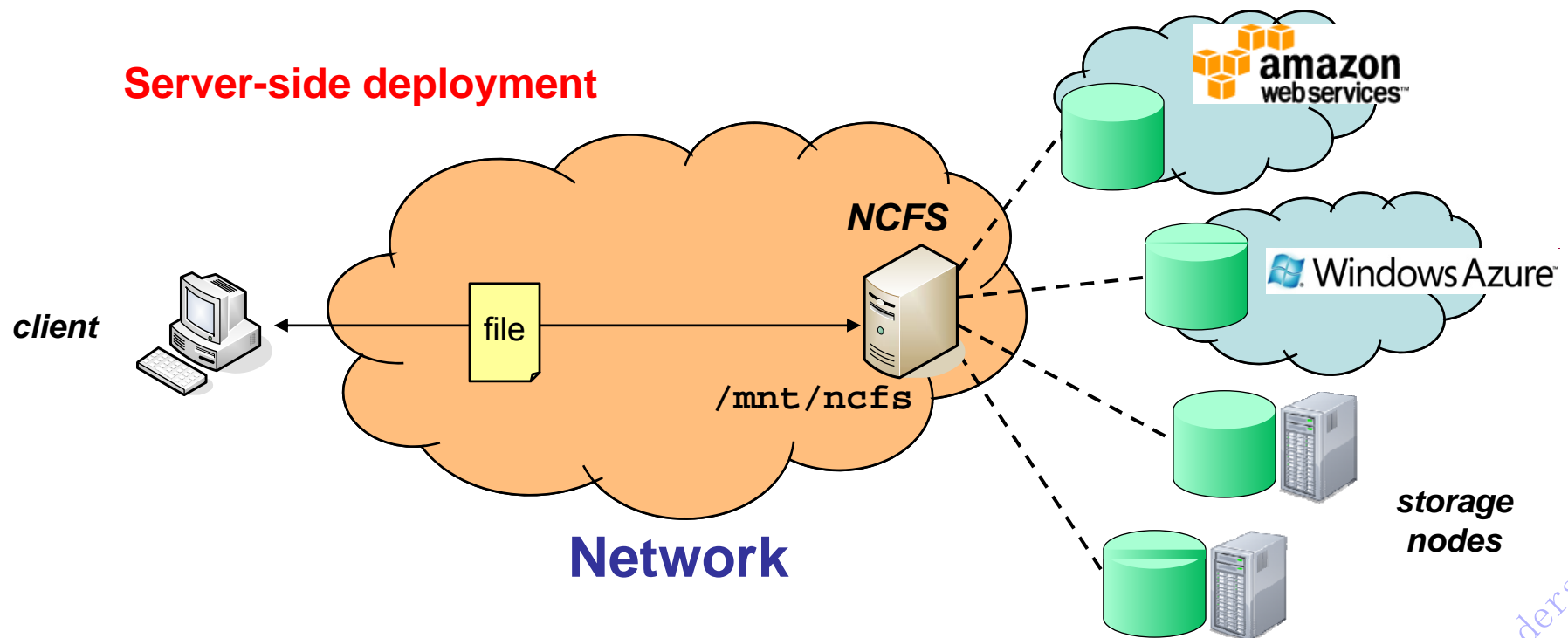
- We consider following MDS codes in this talk:
 - Erasure codes: **RAID**
 - Regenerating codes: **E-MBR($d = n-1$)**
- Both RAID and E-MBR($d=n-1$) codes do not require encoding capabilities in disks

NCFS Architecture

- NCFS is a proxy-based file system that interconnects different types of storage nodes
- Both client-side and server-side deployments are feasible



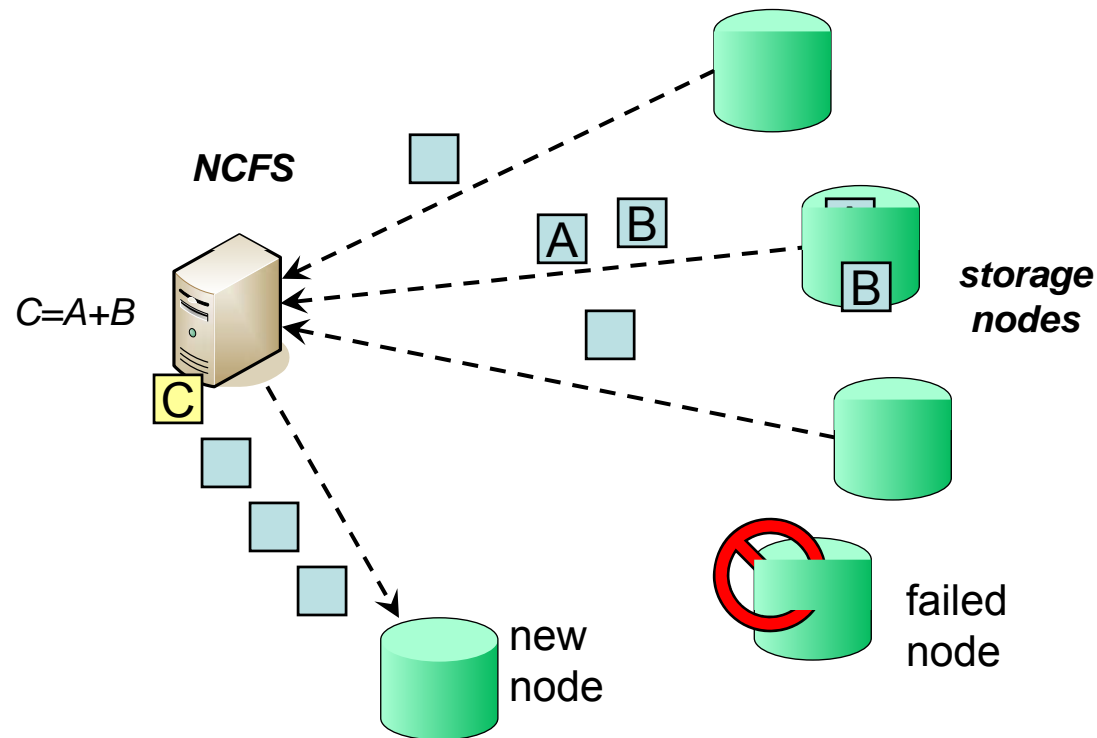
NCFS Architecture



NCFS Architecture

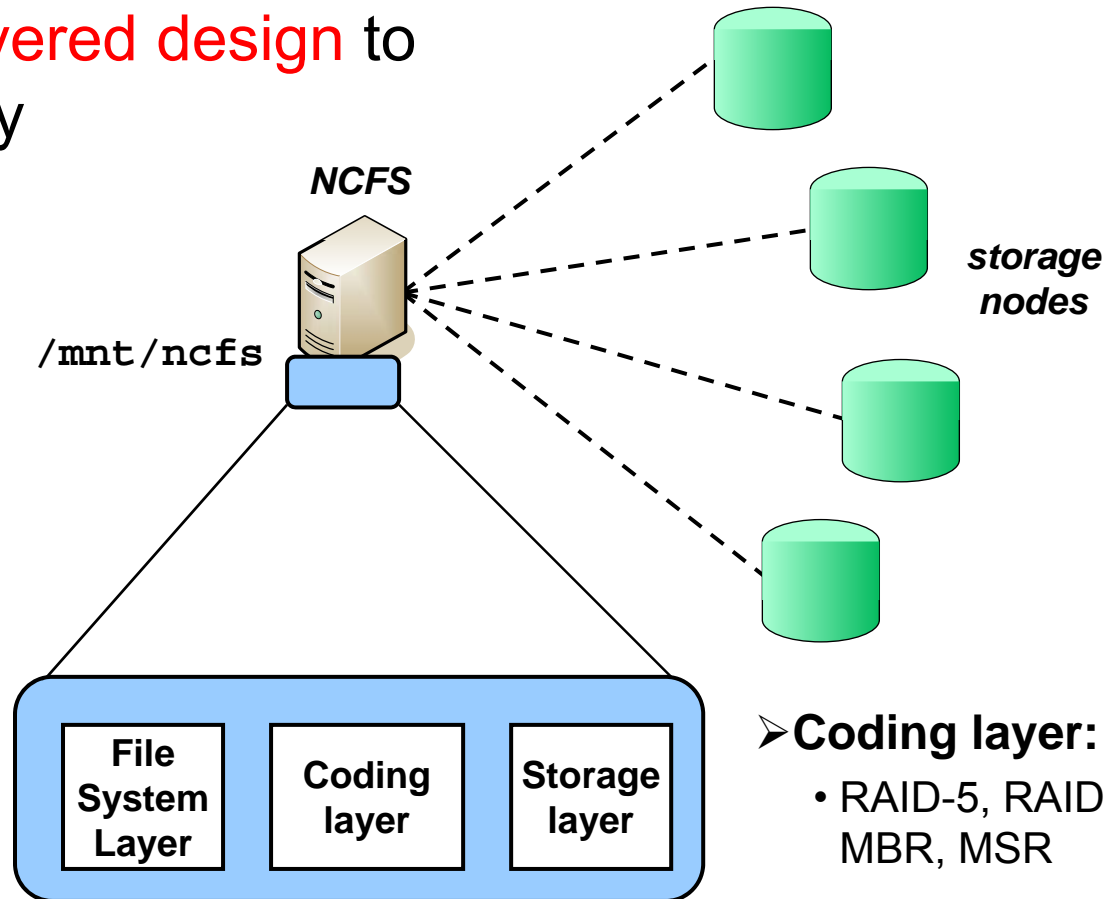
➤ Repair through NCFS:

- read data from survival nodes
- regenerate lost data blocks
- write the regenerated blocks to a new node



Layered Design of NCFS

- NCFS uses a **layered design** to allow extensibility



- **File system layer**

- Implemented in **FUSE**
- Use FUSE to implement all filesystem semantics and repair operations

- **Coding layer:**

- RAID-5, RAID-6, MBR, MSR

- **Storage layer:**

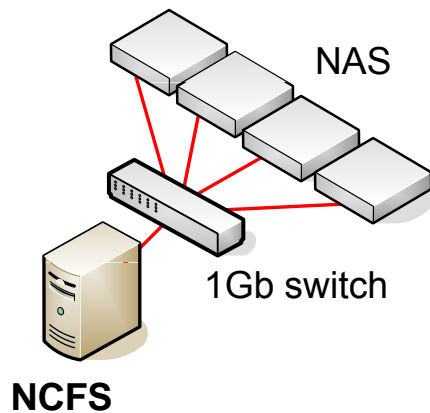
- Interface to storage nodes

Experiments

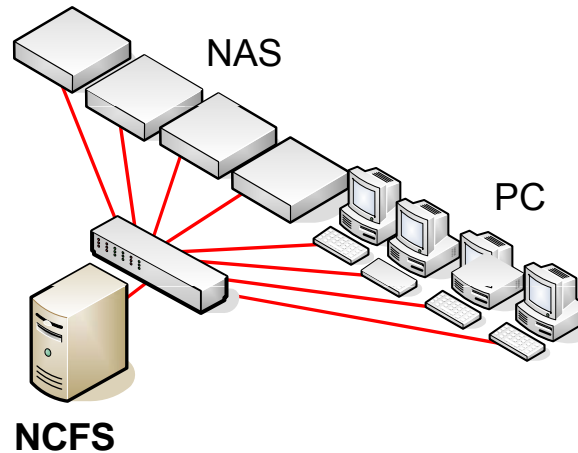
- **Goal:** empirically experiment different coding schemes under NCFS
 - Upload/download throughput
 - Degraded download throughput
 - Repair throughput
- NCFS deployment
 - Intel quad-core machine
 - Linux

Topologies

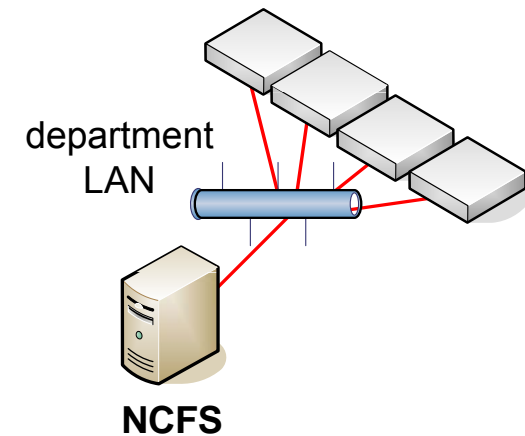
➤ Three network topologies considered:



4-node switch setting
- Test basic functionality



8-node switch setting
- Test scalability

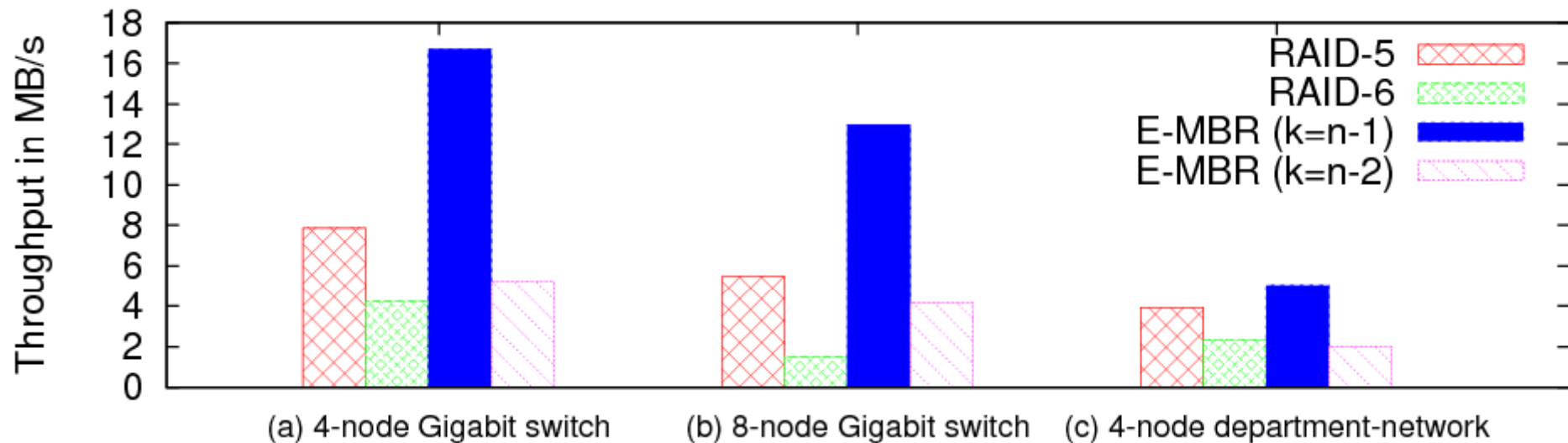


4-node LAN setting
- Test bandwidth bottlenecks

Experiment: Normal Mode

- Normal usage (i.e., all nodes work)
 - Upload a 256-MB file
 - Download a 256-MB file
- Different coding schemes have different storage costs
 - e.g., actual storage for $n = 4$,
 - RAID-5: 341MB
 - RAID-6: 512MB
 - E-MBR($k = n - 1$): 512MB
 - E-MBR($k = n - 2$): 614MB

Normal Upload Throughput



- E-MBR($k=n-1$) has the highest upload throughput because of no code-block update, even it uploads more data

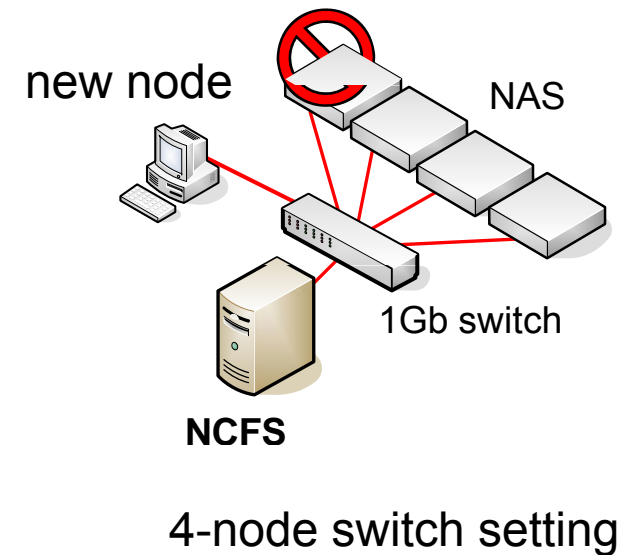
Upload Time Breakdown

4-node switch setting

	Encoding	Disk read	Disk write	Others
RAID-5	9.25%	49.78%	28.64%	12.33%
RAID-6	27.38%	28.94%	36.74%	6.94%
E-MBR(k=n-1)	0.08%	0	83.43%	16.49%
E-MBR(k=n-2)	5.93%	34.30%	52.01%	7.77%

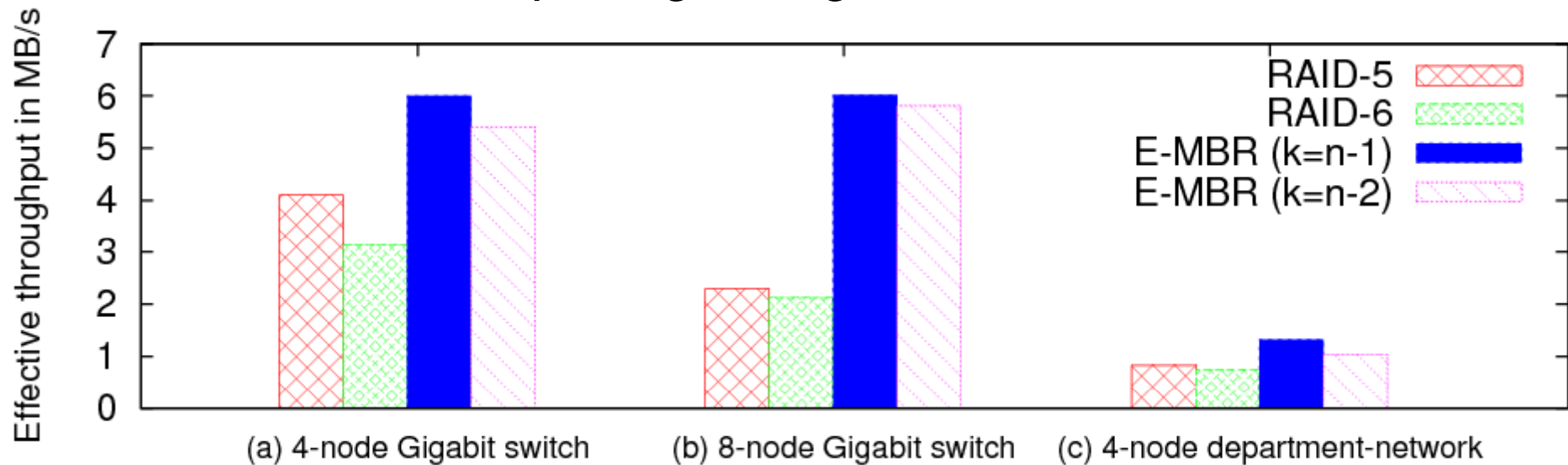
Experiment: Repair

- Recover all lost data to a new node
- Lost data includes native blocks and redundancy
- Measure **effective throughput** T_{eff} :
 - f = fraction of redundant blocks
 - N = size of all lost blocks
 - t = time for recovery
 - $T_{\text{eff}} = (1-f)*N / t$



Repair Throughput

Repairing a single-node failure



- E-MBR has higher repair throughput, which conforms to the theory

Repair Time Breakdown

*Repairing a single-node failure
(4-node switch setting)*

	Encoding	Disk read	Disk write	Others
RAID-5	3.32%	68.62%	27.67%	0.38%
RAID-6	12.68%	57.98%	28.70%	0.64%
E-MBR(k=n-1)	0.096%	46.05%	52.58%	1.27%
E-MBR(k=n-2)	0.95%	46.70%	51.94%	1.26%

Experiment Summary

- Demonstrate how NCFS is used in testing different coding schemes
- E-MBR not only improves repair throughput, but also improves throughput in normal upload/download
 - Trade-off: higher storage overhead
- More coding schemes can be plugged into NCFS

Open Issues

- Performance optimization
 - Caching strategy
 - Grouping multiple block accesses into one request
 - Parallelization
- Security
 - Confidentiality
 - Byzantine fault tolerance
 - Pollution resilience

NCFS References

➤ Source code:

- <http://ansrlab.cse.cuhk.edu.hk/software/ncfs>