Revisiting Network Coding for Warm Blob Storage

Chuang Gan[1], Yuchong Hu[1,2], Leyan Zhao[1], Xin Zhao[1], Pengyu Gong[1], and Dan Feng[1]

Speaker: Ping Chen from Zhejiang University

[1]Huazhong University of Science and Technology
[2]Shenzhen Huazhong University of Science and Technology Research Institute

USENIX FAST 2025

Introduction

- Blob storage
  - Reportedly deployed in Google Cloud, Amazon AWS, Azure, and Facebook
  - Store huge amounts of unstructured data
    - e.g., photos, videos, documents
- Multiple access tiers exist
  - Keep blobs in different performance tiers for the cost-performance trade-off
  - e.g., in Facebook
    - Haystack (**Hot** blob storage system) :
      - stores newly written blobs which are frequently accessed
    - f4 (**Warm** blob storage system) :
      - Haystack's blobs are moved to f4 when they age and become less frequently accessed

Introduction

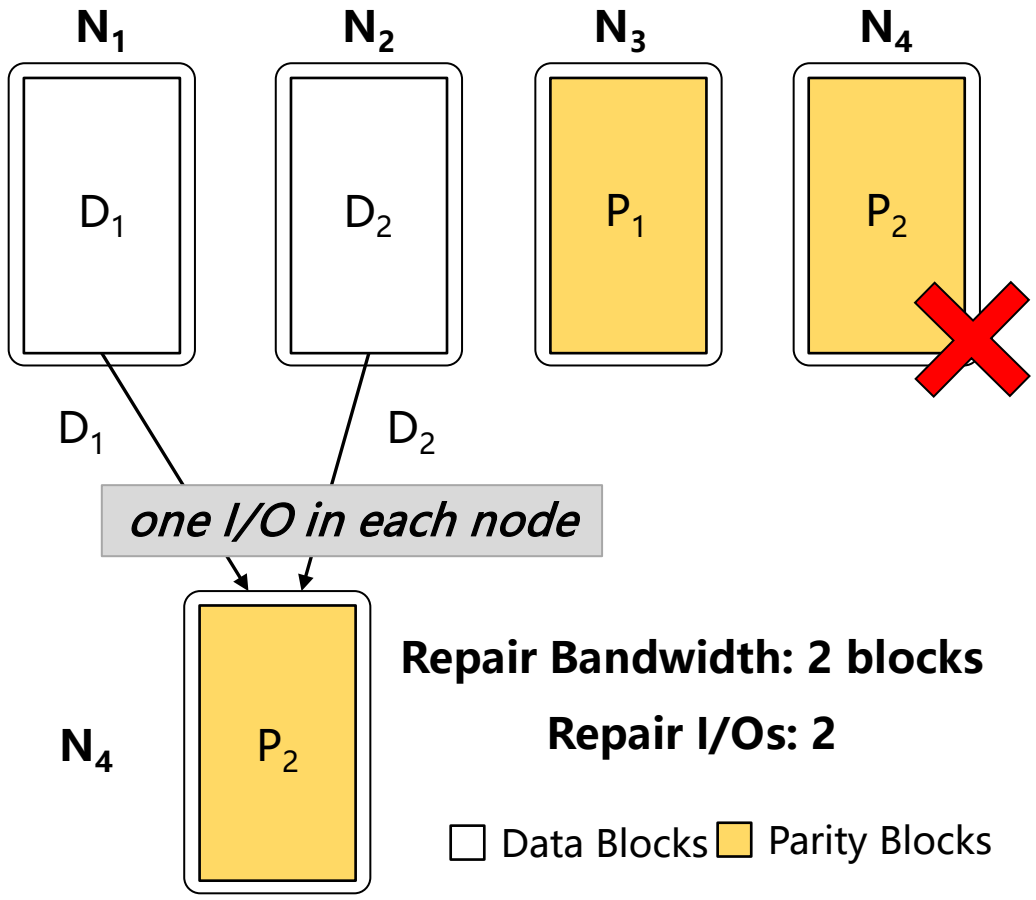
- **Warm blobs** are common
  - Warm blobs in practice account for a majority of blob storage
    - e.g., Facebook's warm blobs account for over 80% of all blobs [Muralidhar, OSDI'14]

- **Small blobs** dominate
  - Blobs in practice have small sizes
  - For example,
    - In Facebook's f4 [Muralidhar, OSDI'14]
      - almost all photo blobs are smaller than 1 MB
      - 60% of video blobs are smaller than 1 MB
    - In Azure [Romero, SoCC'21]
      - 80% of blobs accessed by Azure functions are smaller than 12 KB

Erasure Coding

➢ Erasure coding is often used to provide reliability with low cost

- Reportedly deployed in many blob storage systems (e.g., Facebook f4)
- ***(n,k)*** Stripe: Encode ***k*** data blocks into ***n*** code blocks
- ***MDS***: any *k* of *n* coded blocks in the stripe can recover all data, with minimum redundancy
- Drawback: **high repair cost**



(4,2) Reed-Solomon (RS) codes

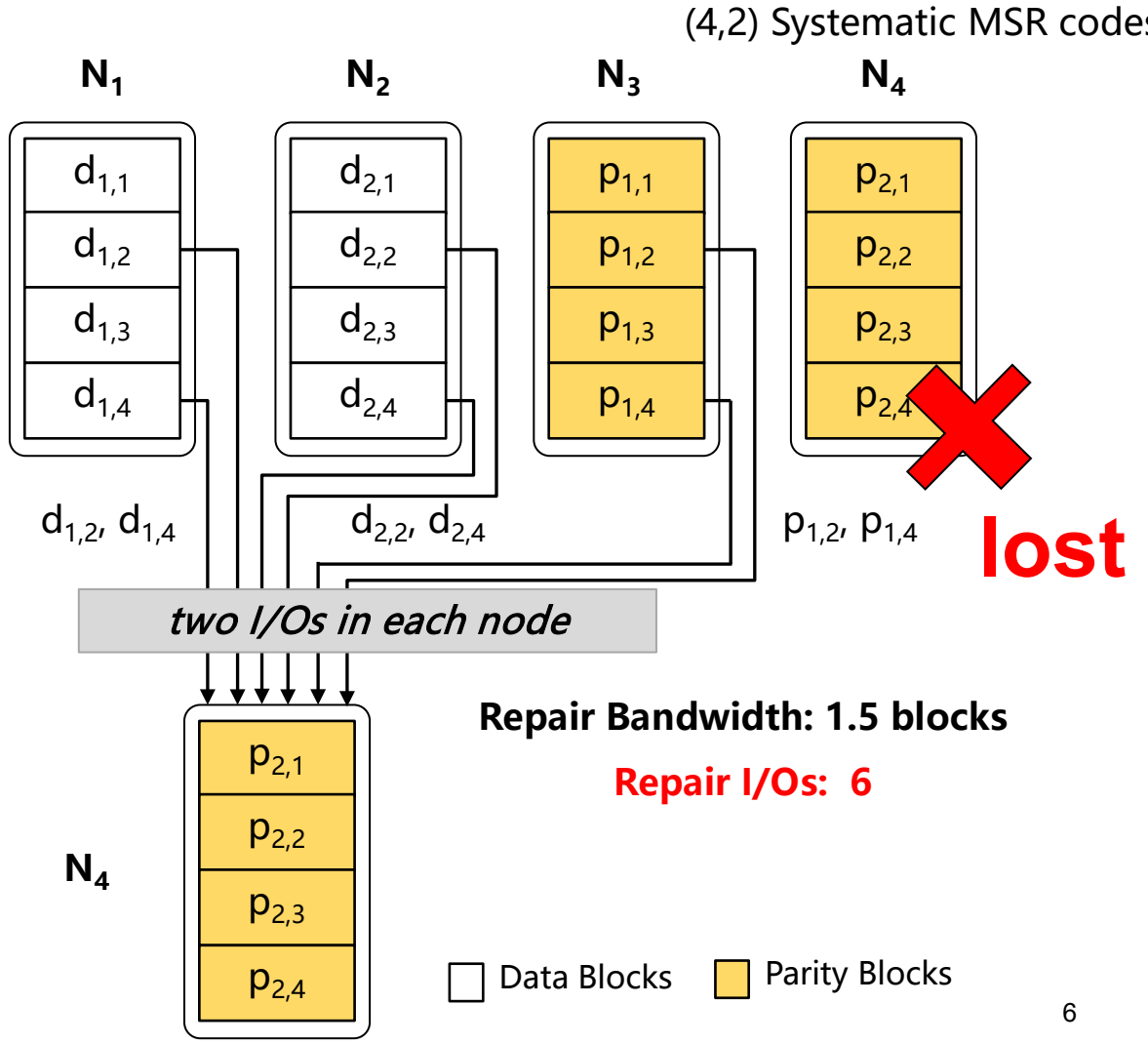
Regenerating Codes

- Regenerating codes are repair-friendly
  - Minimize the amount of data transfers during repair
  - ***Sub-packetization***: each block is divided into numbers of sub-blocks
- Minimum-storage Regenerating (MSR) codes further minimize storage redundancy
  - ***Non-systematic*** MSR codes
    - Store only parity blocks
    - construction of MSR codes [Dimakis, TIT'10]
  - ***Systematic*** MSR codes
    - Store both original data blocks and parity blocks
    - Are widely deployed in industry, e.g., Clay codes [Vajha, FAST'18])

Limitations of Systematic MSR Codes

➢ High sub-packetization level limits actual repair performance

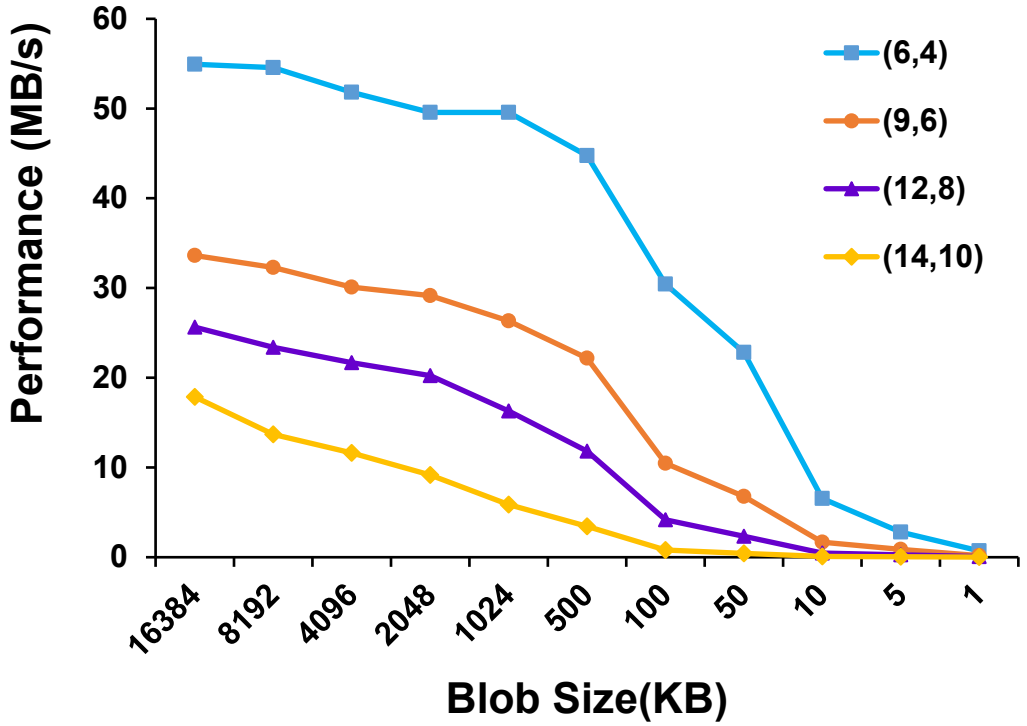
- The number of sub-blocks increases exponentially with (n,k)
  - e.g., a single block of (14,10) Clay code is divided into 256 sub-blocks
- A high number of sub-blocks incurs high non-contiguous I/Os
  - e.g., up to 64 non-contiguous I/Os for repairing a single lost block of (14,10) Clay code [Vajha, FAST'18]



Limitations of Systematic MSR Codes

➢ *Limitation #1*: Slow single-block repair, particularly for small blobs.

- A small blob is divided into smaller blocks
  - e.g., for 1MB blob encoded via the (14,10) Clay code, each block is 1MB/10 ≈ 100KB
- Slow single-block repair on small blobs encoded by Clay codes



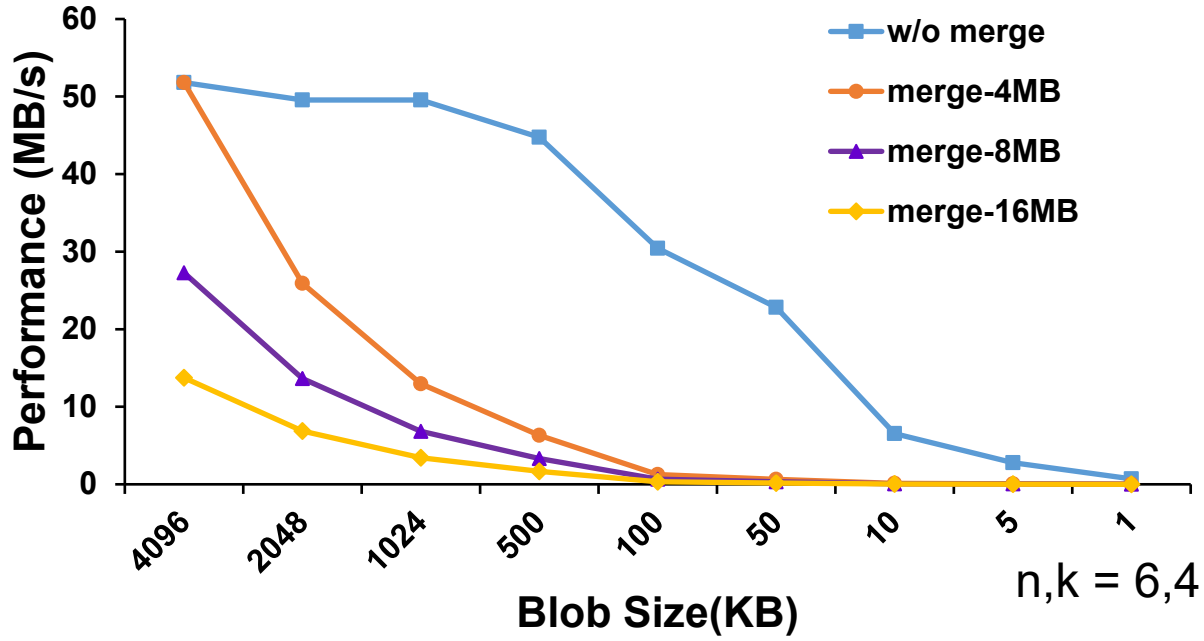
Repair performance drops dramatically as the blob size decreases from 1 MB to 10 KB

repair performance of different (n,k) Clay codes for various size blobs

Limitations of Systematic MSR Codes

➢ *Limitation #2*: Slow degraded read for small blobs

- Contiguous layout (Facebook f4)
  - Merge multiple small blobs into the equal-sized large blocks
- Degraded read amplification
  - Clay codes repair data with the granularity of a block

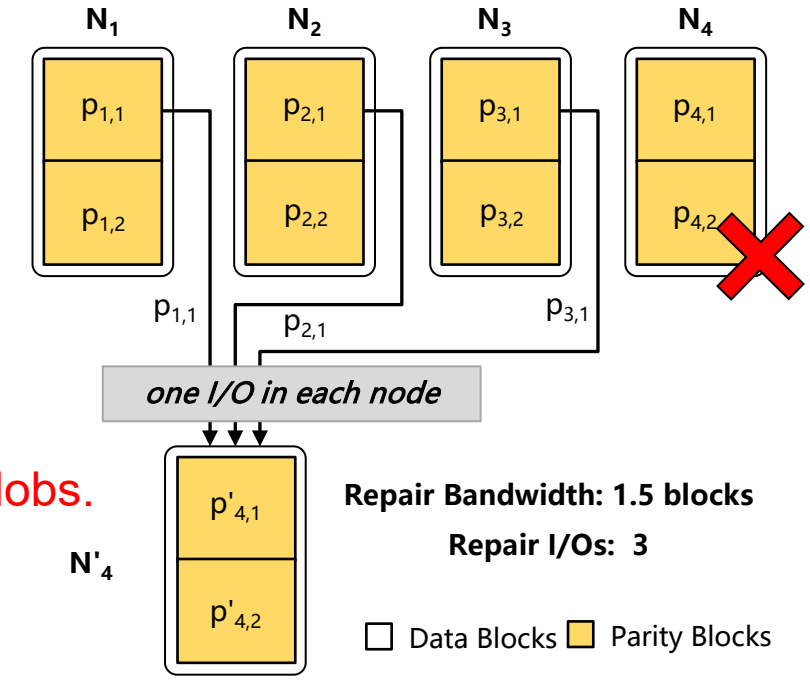


Degraded read performance drops significantly as the merge size increases from 4MB to 16MB

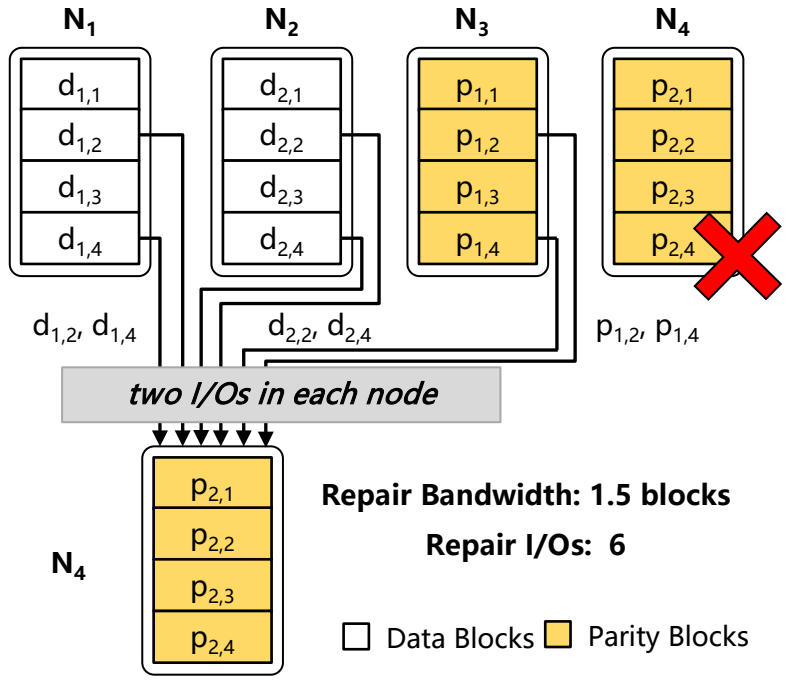
degrade read performance of (6,4) Clay code for various size blobs under different merging granularity

Motivation

- Potential of non-systematic MSR codes
  - Achieve repair-optimal with low sub-packetization level (fewer sub-blocks)
  - Require fewer sub-blocks within a block to repair the lost block (fewer I/Os)



(a) Non-systematic MSR codes (F-MSR [Hu, FAST'12])



(b) Systematic MSR codes (Clay codes [Vajha, FAST'18])

friendly to small blobs.

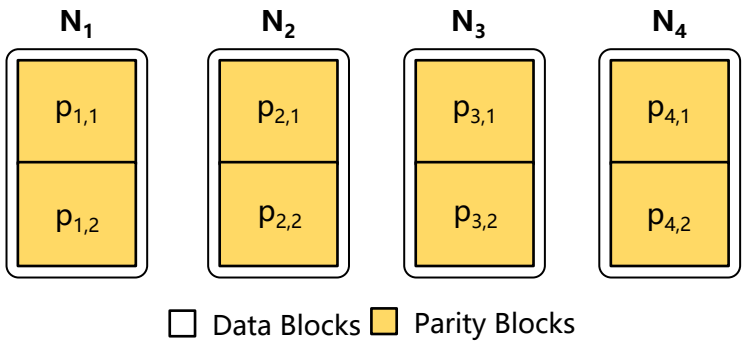
Low sub-packetization level is friendly to small blobs!

Our Deisgn

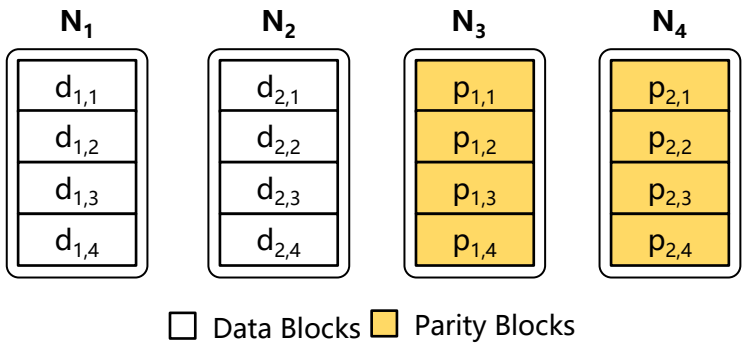
- **NCBlob:** design of a hybrid MSR-based warm blob storage system
  - Reduced read amplification via **locality-based encoding**
  - Efficient repair via a **hybrid MSR codes scheme**
  - Generalized parameters via a **rotation-based sub-block selection scheme**

Locality-based encoding

- *Challenge #1*: Poor read performance of non-systematic MSR codes
  - Do not store original data blocks
  - Read amplification
    - Reading one data block has to decode *k* blocks from other nodes
      - In F-MSR (Figure (a)), read any byte of original data → decode 2 blocks in the stripe
      - In Clay codes (Figure (b)), original data can be directly read out



(a) Non-systematic MSR codes (F-MSR [Hu, FAST'12])



(b) Systematic MSR codes (Clay codes [Vajha, FAST'18])

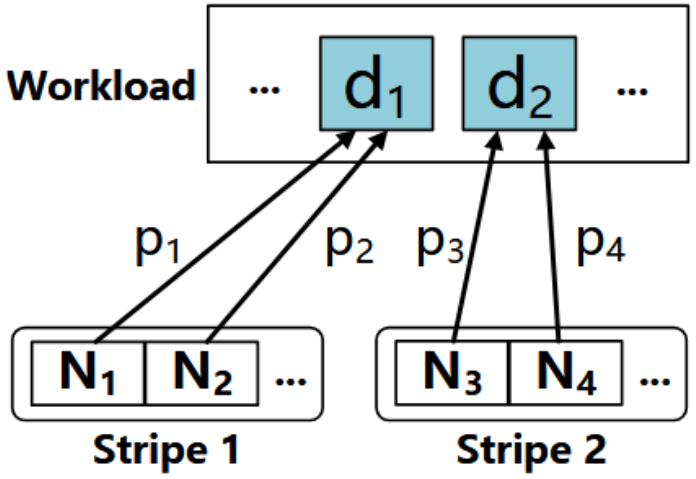
Locality-based encoding

- Insight
  - Access locality can be leveraged to optimize access performance
    - If any bytes from the group of data with access locality is accessed, there is a good chance that other data in the group will also be accessed [Annamalai, OSDI'18]
- Blobs exhibit good access locality
  - Intra-blob locality [Romero, SoCC'21]
    - Small blob itself is commonly accessed as a whole
      - If any bytes from such a blob is read, the rest of it should be pre-fetched
  - Inter-blob locality [Hendricks, CMU Technical Report 2006]
    - Multiple some blobs are often accessed together
      - Blobs sharing the same attributes (e.g., user ID, app ID, or region [*]) in their metadata

[*] Microsoft. Azure blob storage - access tiers. https://learn.microsoft.com/en-us/azure/ storage/blobs/access-tiers-overview

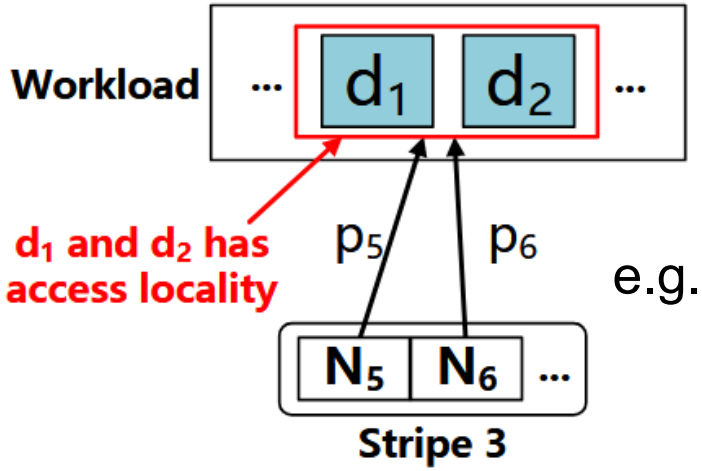
Locality-based encoding

- Design idea: Performing encoding on the group of data with access locality
  - Divide the group of data with access locality into *k* data blocks
  - Non-systematically encode the data blocks into *n* parity blocks
  - Reduce read amplification



(a) Read w/o considering locality

Read bandwidth: 4 blocks



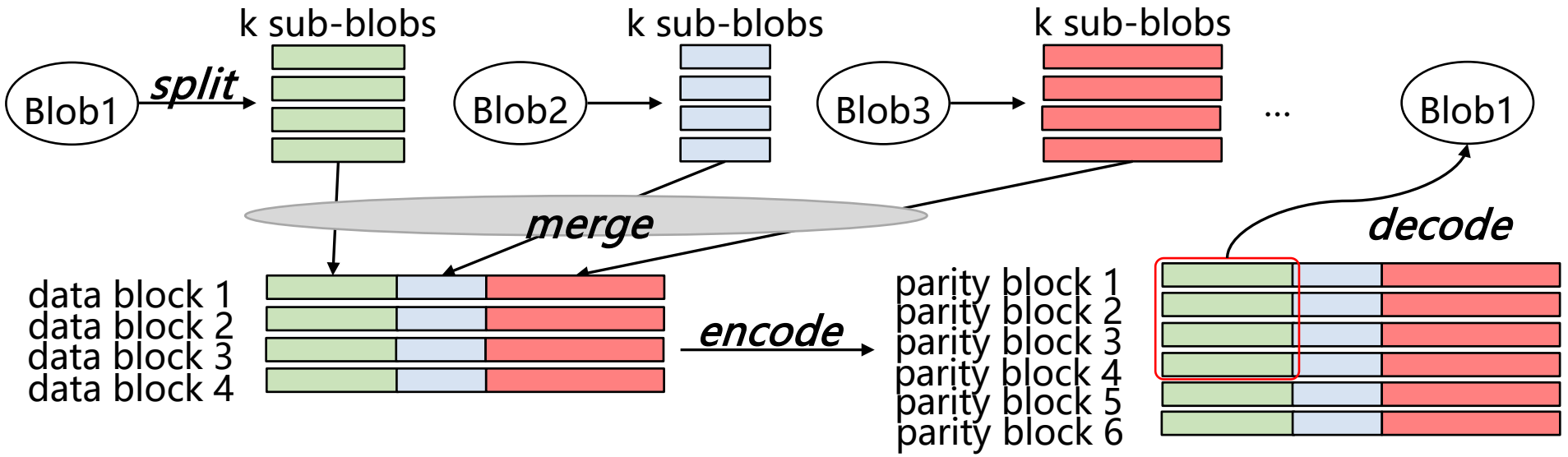
(b) Read w/ considering locality

Read bandwidth: 2 blocks

e.g., d_1 and d_2 has access locality; when d_1 is read, **d_2 has a good chance to be read immediately**

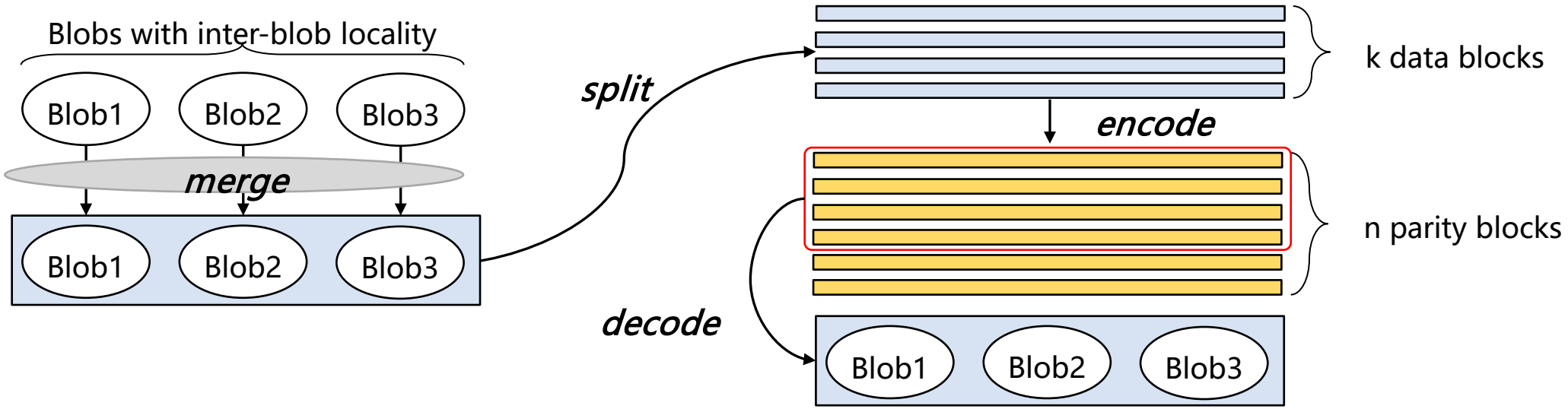
Locality-based encoding

- Split-merge-encode:
  - Based on **intra-blob locality**, NCBlob encode a blob into a stripe
    - Split: each of the small blobs are split into *k* equal-sized sub-blobs
    - Merge: merge the sub-blobs of multiple blobs into *k* data blocks
    - Encode: encode *k* data blocks into *n* parity blocks
  - No read amplification
    - reading Blob1 only transfers the size of Blob1



Locality-based encoding

- Merge-split-encode:
  - Based on **inter-blob locality**, NCBlob encode the group of blobs with access locality into a stripe
    - Merge: merge multiple small blobs with inter-blob locality into a fixed-size group
    - Split: split the group into *k* data blocks
    - Encode: encode *k* data blocks into *n* parity blocks
  - Reduce read amplification



Hybrid MSR codes

➢ *Challenge #2*: How to efficiently repair and read large blobs?

Design idea:

Small blobs: with non-systematic MSR codes

Large blobs: with Clay codes

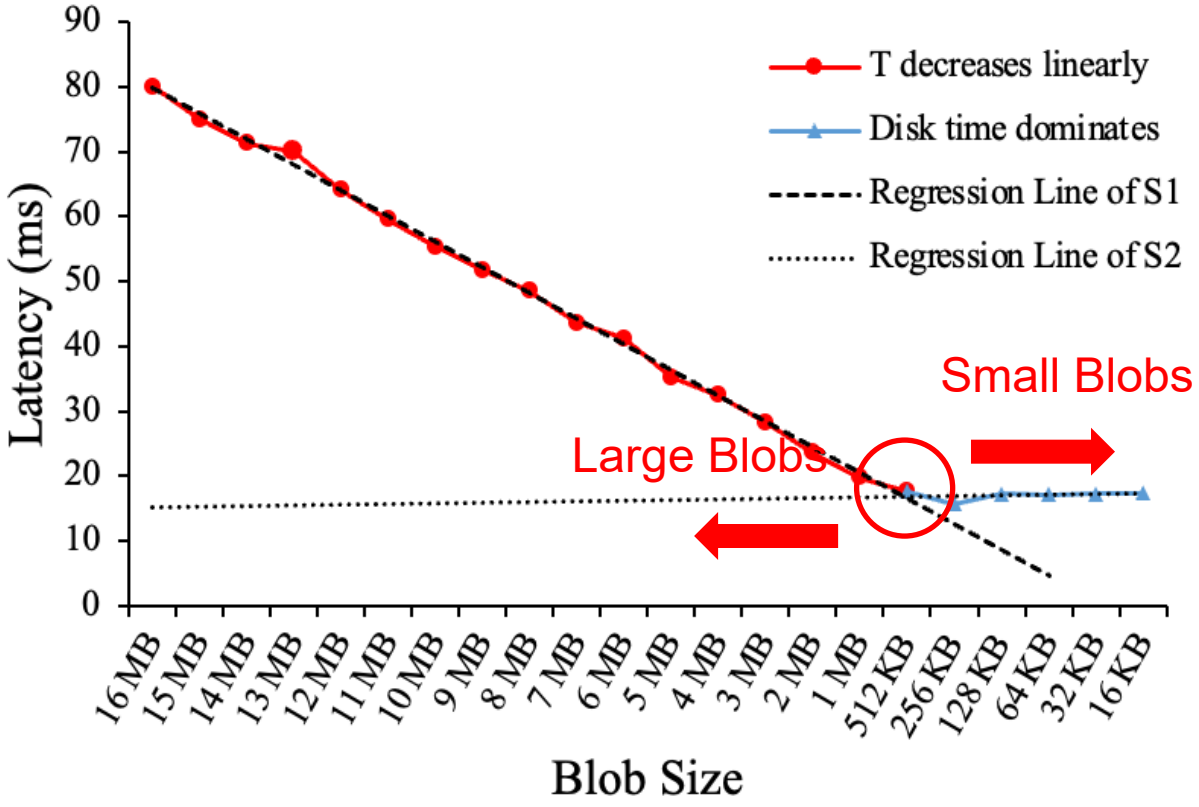
Design idea:

Model the single-block repair time *T* of Clay codes for differentiating blobs

T increases linearly

T can be dominated by the disk I/O time

Select the blob size when T dominates



Rotation-based sub-block selection

- *Challenge #3*: State-of-the-art non-systematic MSR codes only support *n-k=2*
- Design idea: select the sub-block in rotation for each iterative repair
- Prove that the rotation-based sub-block selection can support non-systematic MSR codes with *n-k=3*
- Simulate that the rotation-based sub-block selection can support non-systematic MSR codes with *n-k=4* under $GF(2^{16})$
  - *(14,10)* NCBlob can still work after more than *10K* times of repair iterations
    - Considering a node failure with an average lifetime of 4 years[Sathiamoorthy, VLDB'13], *(14,10)* NCBlob can sustain a lifetime of about 2857 years

Detailed proof can be found in the appendix of the paper

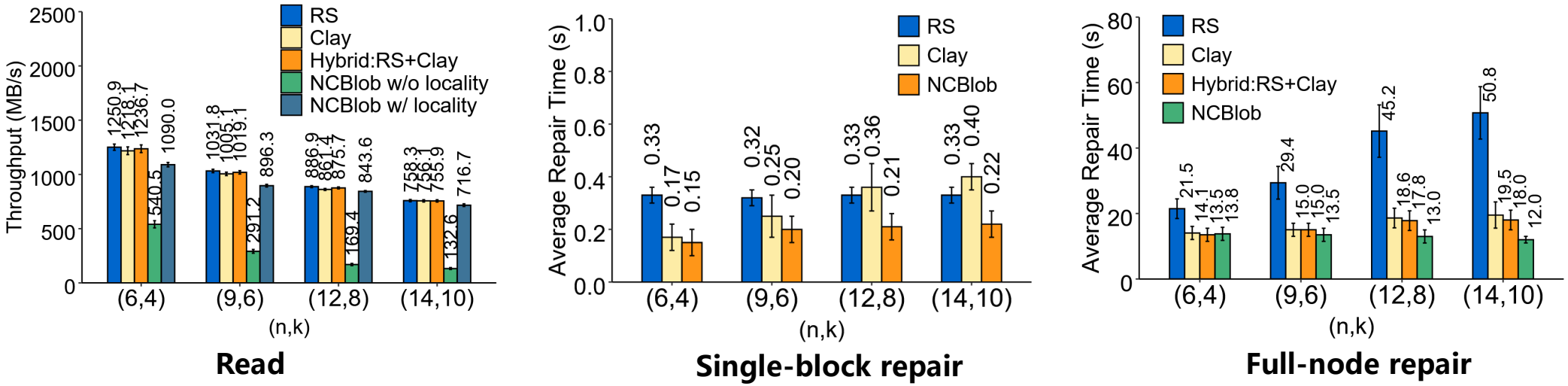
Implementation

- Architecture:
  - Metadata server
  - Data server
  - Coordinator: assign read and repair tasks
  - 18.6 K SLoC in C++
- Modules:
  - Locality management
  - Coding
    - Based on the coding repository of Ceph[*] and ParaRC[Li, FAST'23]
  - Distribution
    - Evenly distribute data over multiple nodes based on the Placement Group (PG) in Ceph [*]
  - Repair
    - Parallel repair and read for non-systematic MSR codes

[*] Ceph - Erasure code. https://docs.ceph.com/en/ latest/rados/operations/erasure-code.

Evaluation

- NCBlob achieves comparable read throughput with Clay codes
  - Only 2.1% lower than Clay codes for (n,k)=(12,8)
  - Experiments conducted on the Azure Function Blob dataset[*]
- NCBlob achieves efficient repair compared with Clay codes
  - Single-block repair time reduced by up to 45.0%
  - Full-node repair time reduced by up to 38.4%



More experiments on NCBlob in the paper

[*] Azure. Azure functions blob dataset. https:// github.com/Azure/AzurePublicDataset.

Conclusions

- Analyze the limitations of systematic MSR codes in handling small blobs
- Propose locality-based coding to apply non-systematic MSR codes on small blobs for fast repair
- Design NCBlob, an efficient hybrid MSR-based warm blob storage system that supports generalized parameters
- Implement NCBlob and conduct experiments to show NCBlob's efficiency
- Prototype:
  - **https://github.com/YuchongHu/NCBlob**

Conclusions

Contact chelgan@hust.edu.cn for any questions

Chuang Gan, Huazhong University of Science and Technology