StripeMerge: Efficient Wide-Stripe Generation for Large-Scale Erasure-Coded Storage

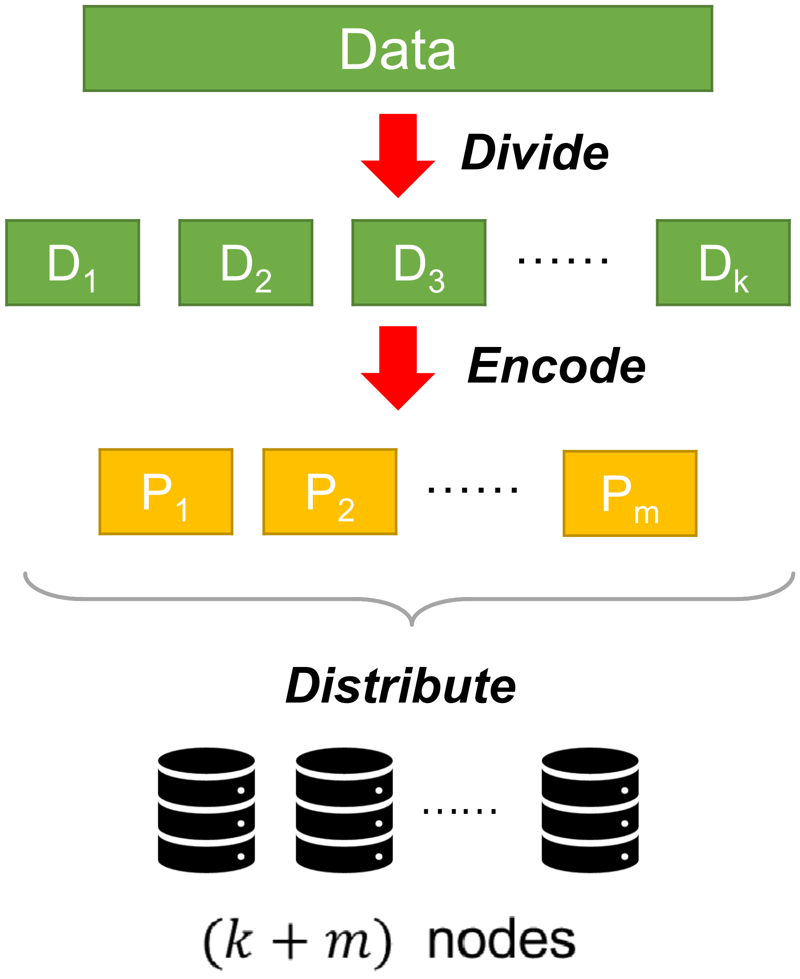
Qiaori Yao[1], Yuchong Hu[1], Liangfeng Cheng[1], Patrick P. C. Lee[2], Dan Feng[1], Weichun Wang[3], Wei Chen[3]

[1] *Huazhong University of Science and Technology*
[2] *The Chinese University of Hong Kong*
[3] *HIKVISION*

Erasure Coding

- A widely adopted redundancy technique
  - An alternative to replication
  - **Low-cost** fault tolerance

- Reed-Solomon (RS) codes
  - (k, m): k data chunks $\xrightarrow{\text{encoding matrix}}$ m parity chunks
  - Stripe: $k + m$ chunks, stored in $k + m$ nodes
  - Redundancy: $\frac{k+m}{k}$



Wide-stripe Erasure Coding

- Wide stripes:
  - Goal: extreme storage savings
  - Definition: very large k, small m ; redundancy: $\frac{k+m}{k} \rightarrow 1$
  - Our previous work: ***ECWide*** [FAST'21]
- How to generate a wide stripe?
  - Natural idea: **direct generation**
    - Expensive repair: retrieve k chunks to repair one chunk
  - Our idea: **tiered generation**
    - Motivation: access frequency is high at first, but decreases as data age

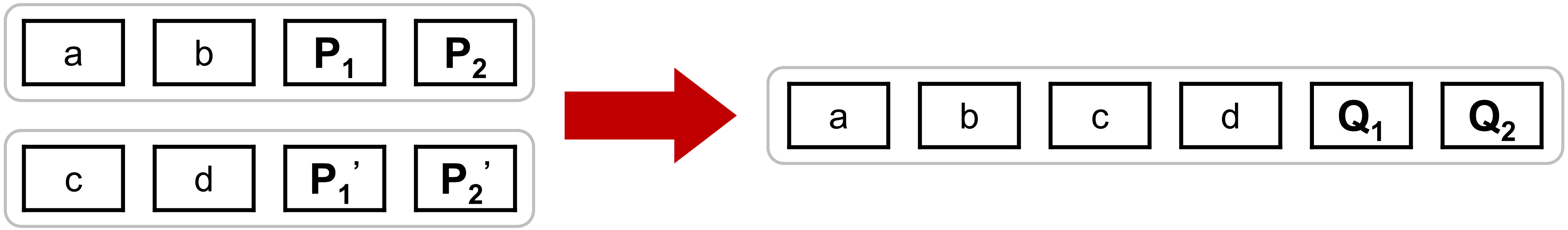
Original data → Narrow stripes (small k) —**Re-encode**→ Wide stripes (large k)

High access frequency

Low access frequency

Problem

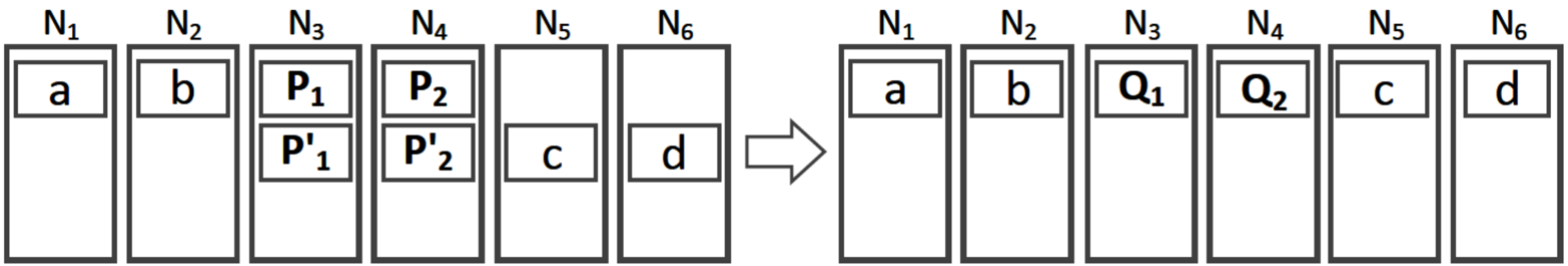
- Re-encoding in tiered generation
  1. Relocate data chunks
  2. Regenerate parity chunks
- Challenge
  - Substantial bandwidth overhead in data transfers
  - **How to mitigate data transfers during wide-stripe generation?**
- Problem: Two (k, m) stripes **merge** into a $(2k, m)$ stripe



Perfect Merging

- Generation without any transfer
  - Idea: both data and parity chunks are **locally generated**
  - Definition of Perfect Merging:
    1. Data chunks reside in different nodes
    2. Parity chunks have identical encoding coefficients and reside in the same nodes
  - Insight: **Vandermonde-based RS codes** allow local generation of new parity chunks, by using old parity chunks as input

$$\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \end{bmatrix} + \begin{bmatrix} 1^2 \cdot P'_1 \\ 2^2 \cdot P'_2 \end{bmatrix}.$$



Our Contributions

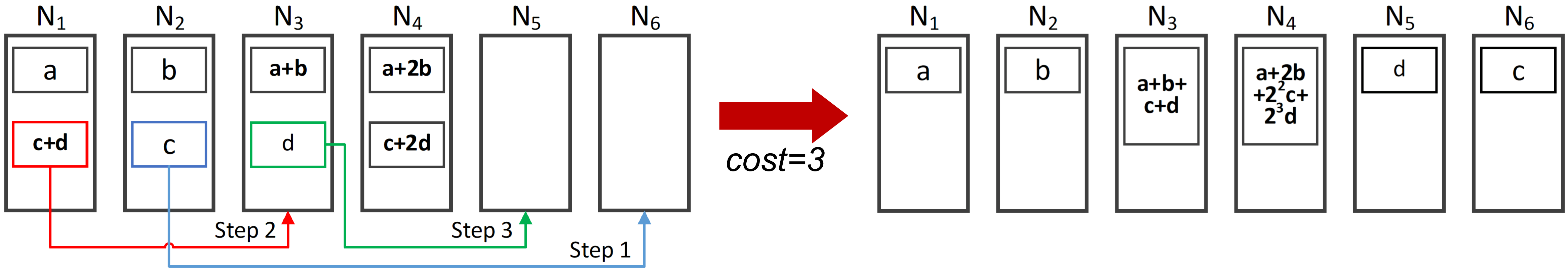
- The first to address the wide-stripe generation problem
- Model:
  - Formulate this problem with bipartite graph model
  - Prove the existence of an optimal scheme that exploits the perfect merging property, but it has prohibitive algorithmic complexity
- Algorithm: **StripeMerge**
  a) A greedy heuristic algorithm that reduces the algorithmic complexity
  b) A parity-aligned heuristic algorithm that further enhances the former
- Evaluation:
  - Significantly reduces data transfers for wide stripe generation by up to 87.8% over a state-of-the-art storage scaling approach

Bipartite Graph Model

- Formulate the problem
  - Background: a large-scale storage system with N nodes, **sufficiently large** number of (k, m) narrow stripes, randomly placed chunks
  - Goal: select all pairs of narrow stripes that satisfy perfect merging
  - Model: **bipartite graph** *(see details in the paper)*
- Existence: Theorem 1
  - Conclusion: when the number of narrow stripes is sufficiently large, 0-cost merging scheme always exists theoretically. *(see details in the paper)*
- Infeasibility in practice
  - High algorithmic complexity: $O(n^{2.5})$, maximum matching problem on a bipartite graph
  - A large number of stripes required: only a limited number of stripes in practice

StripeMerge-G

- Naive greedy heuristic
  - Idea: transfer chunks to satisfy perfect merging
  - Merging cost: the number of transferred chunks
  - Algorithm:
    1. Get merging costs of **all pairs**;
    2. Select the **minimal** pair of stripes every time
  - Time complexity: $O((k+m)n^2)$; still time-consuming



StripeMerge-P

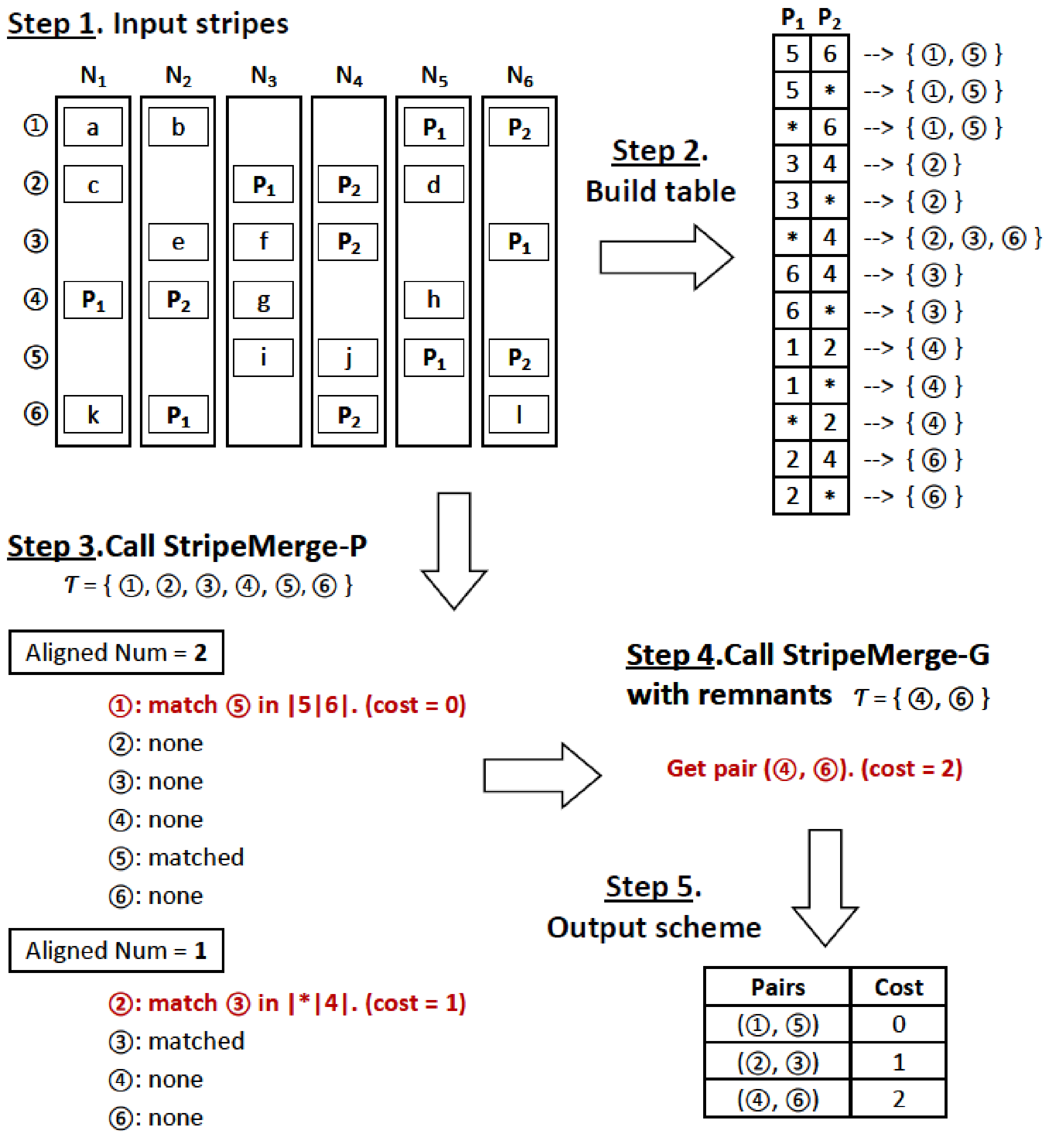
- Parity-aligned heuristic
  - Main idea:
    - **Parity-aligned**: parity chunks have identical encoding coefficients and reside in the same nodes
    - Search in parity-aligned sets, in order to **rapidly** merge a large number of stripes
  - Hash table : accelerate the construction of parity-aligned sets
  - Algorithm:
    1. Search for pairs in parity-aligned sets
    2. Select the minimal one in 1. every time
    3. Use *StripeMerge-G* to deal with remaining stripes

    (see details in the paper)
  - **Time complexity**: $O((k+m)mn)$ in the best cases

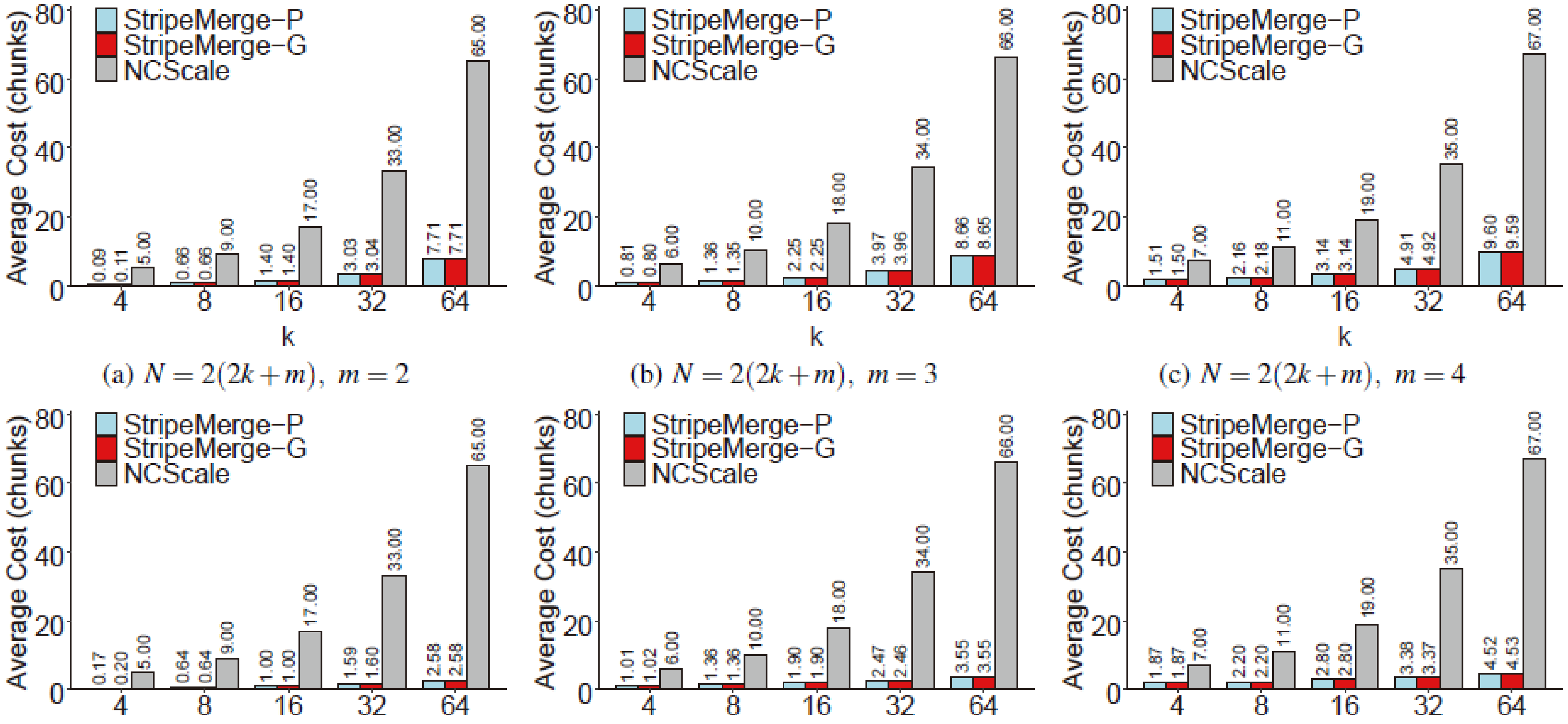
Example

- Example of StripeMerge-P
  1. Get the stripes
  2. Build the hash table
  3. Call StripeMerge-P
  4. Call StripeMerge-G to deal with remaining stripes
  5. Get the scheme of merging narrow stripes into wide stripes

  (see details in the paper)



Evaluation - Simulations



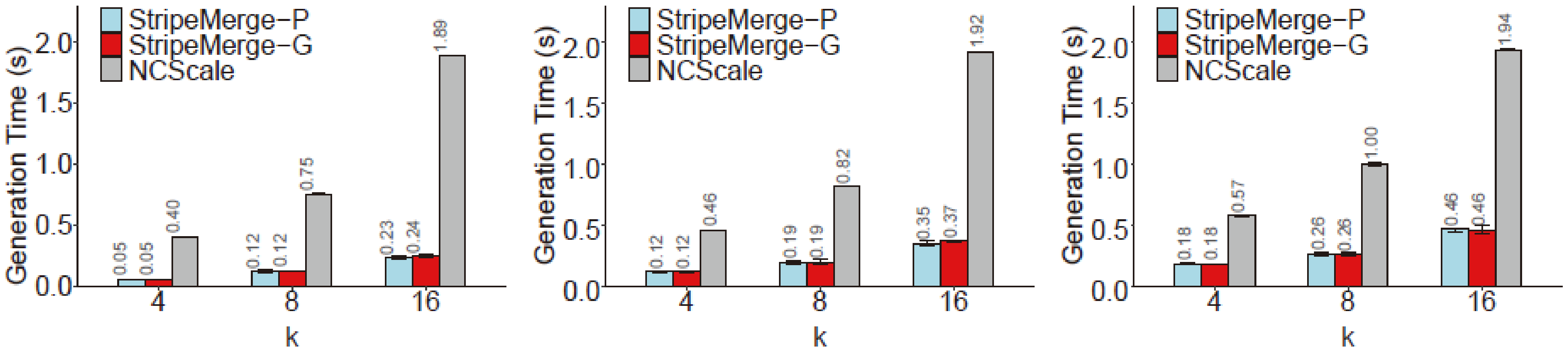
(a) $N = 2(2k+m),\ m = 2$ (b) $N = 2(2k+m),\ m = 3$ (c) $N = 2(2k+m),\ m = 4$

(d) $N = 4(2k+m),\ m = 2$ (e) $N = 4(2k+m),\ m = 3$ (f) $N = 4(2k+m),\ m = 4$

(see more in the paper)

- StripeMerge significantly reduces the wide-stripe generation bandwidth of the state-of-the-art storage scaling approach in all cases, up to **96%**.

Evaluation - Experiments



(a) $m = 2$ (b) $m = 3$ (c) $m = 4$

(see more in the paper)

- StripeMerge significantly reduces the overall wide-stripe generation time of the state-of-the-art storage scaling approach under the same parameters of (k, m) , up to **87.8%**.

Conclusions

- Propose StripeMerge, a novel mechanism that merges narrow stripes to efficiently generate wide stripes for large-scale erasure-coded storage
- Prove the existence of an optimal scheme for wide-stripe generation via bipartite graph modeling
- Two practical heuristics to realize efficient wide-stripe generation
- Evaluations demonstrate the wide-stripe generation efficiency of StripeMerge over state-of-the-arts

Source code: https://github.com/yuchonghu/stripe-merge

THANK YOU

Contacts:

Yuchong Hu yuchonghu@hust.edu.cn

Patrick Lee pclee@cse.cuhk.edu.hk

Qiaori Yao yaoqr@hust.edu.cn