Exploiting Combined Locality for Wide-Stripe Erasure Coding in Distributed Storage

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

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

Speaker: **Yuchong Hu**

Erasure Coding

- Low-cost redundancy while maintaining availability
  - Parameters: n and k
  - Stripe: k data chunks and n-k parity chunks
  - Redundancy: n/k

- State-of-the-art erasure coding
  - Parameters: n ≤ 20, n-k = 3 or 4
  - Stripe: medium range
  - Redundancy: from 1.18 to 1.50

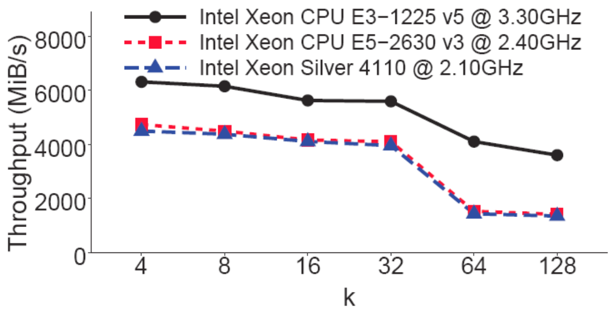
Storage systems	(n,k)	**Redundancy**
Google Colossus [25]	(9,6)	1.50
Quantcast File System [49]	(9,6)	1.50
Hadoop Distributed File System [3]	(9,6)	1.50
Baidu Atlas [36]	(12,8)	1.50
Facebook f4 [47]	(14,10)	1.40
Yahoo Cloud Object Store [48]	(11,8)	1.38
Windows Azure Storage [34]	(16,12)	1.33
Tencent Ultra-Cold Storage [8]	(12,10)	1.20
Pelican [12]	(18,15)	1.20
Backblaze Vaults [13]	(20,17)	1.18

Wide-stripe Erasure Coding

- Motivation
  - Can we further reduce redundancy?
  - Small redundancy reduction (e.g., from 1.5 to 1.33) can save millions of dollars in production [Plank and Huang, FAST'13]
- Wide stripes
  - Parameters: n and k are very large while n-k = 3 or 4.
  - Redundancy: n/k → 1 (near-optimal)
  - Goal: Extreme storage savings
  - Example: VAST considers (n,k) = (154,150) with redundancy = 1.027

Challenges for Wide Stripes

- Expensive repair
  - (n,k) RS code repairs a chunk by retrieving k chunks
  - Large k in wide stripes → more bandwidth and I/O
- Expensive encoding
  - Limited CPU cache: as k increases, more difficult to fit wide stripes in cache



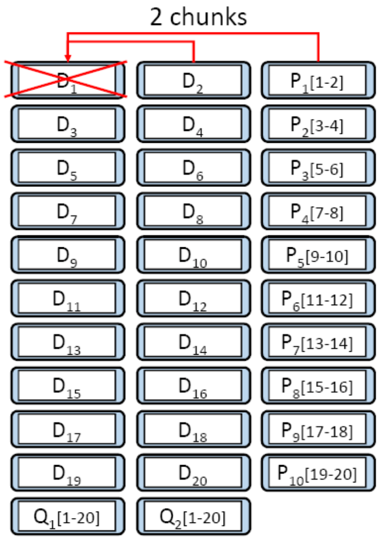
- Expensive updates
  - Same as in traditional stripes: any updated data chunk causes all n-k parity chunks to be updated

Locality in Erasure-coded Repair

- Parity locality
  - Locally repairable codes (LRCs): (n,k,r) Azure-LRC [Huang. ATC'12]
  - **Reducing repair penalty**: encodes every r data chunks into a parity chunk, so repairing a lost chunk only accesses r **local** chunks (r < k)

Repair bandwidth: 2 chunks

Redundancy: 1.6

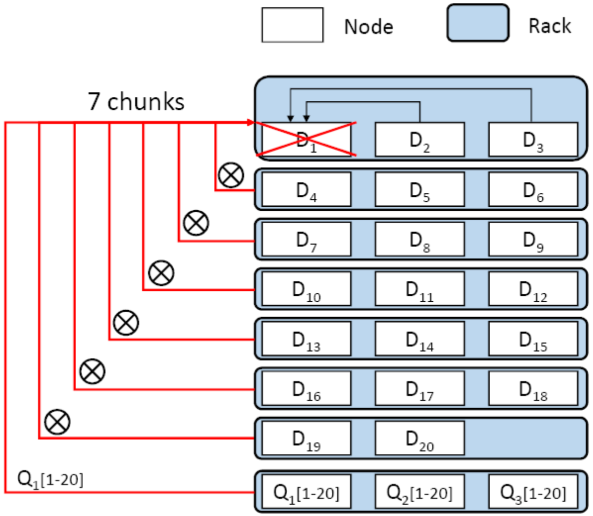


Parity locality: (32, 20, 2) Azure-LRC

Locality in Erasure-coded Repair

➢ Topology locality

- (n,k) RS-coded chunks placed in z racks: (n,k,z) TL
- **Reducing cross-rack repair bandwidth**: splits a repair operation into **local** inner-rack repair and cross-rack repair sub-operations



Topology locality: (23, 20, 8) TL

Cross-rack repair bandwidth: 7 chunks

Redundancy: 1.15 (optimal)

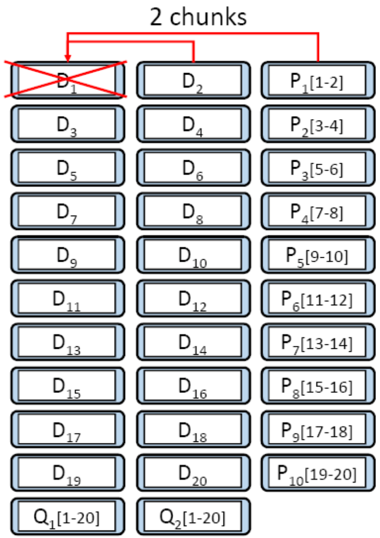
Existing locality schemes for wide stripes

➢ Trade-off between redundancy and repair penalty

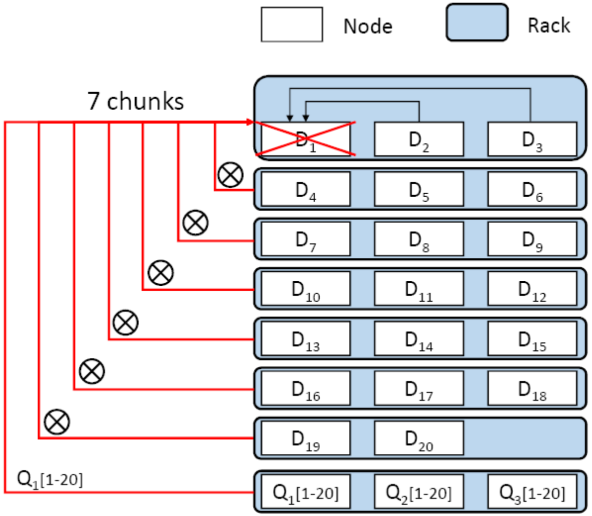
- Parity locality incurs **high redundancy**
- Topology locality incurs **high cross-rack repair bandwidth**

Redundancy: 1.6

Cross-rack repair bandwidth: 2 chunks



Parity locality: (32, 20, 2) Azure-LRC



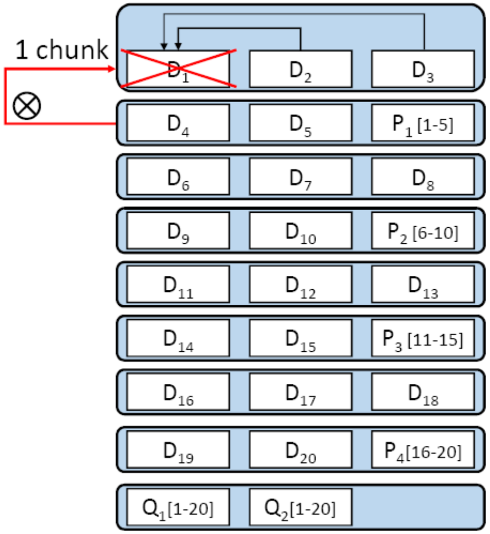
Topology locality: (23, 20, 8) TL

Cross-rack repair bandwidth: 7 chunks

Redundancy: 1.15 (optimal)

Motivating Example

- Combined Locality: (n,k,r,z) CL
  - Idea: combine parity locality and topology locality for better trade-off
  - Example: (26,20,5,9) CL = (26,20,5) Azure-LRC placed in 9 racks



Combined locality: (26, 20, 5, 9) CL

- Cross-rack repair bandwidth: only one chunk
  - less than TL (7 chunks)
  - less than LRC (2 chunks)
- The redundancy: 1.3
  - lower than LRC (1.6)
  - closer to TL (1.15)

Our Contributions

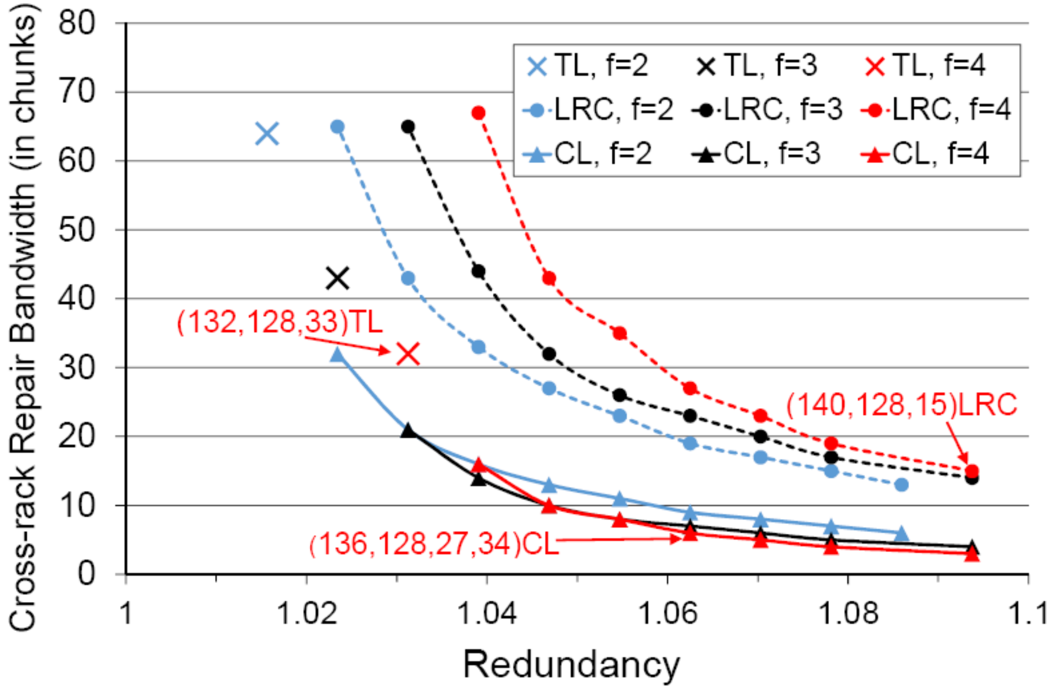
- First systematic study on wide-stripe repair problem
  - Construction details of **combined locality**
  - Trade-off analysis between redundancy and cross-rack repair bandwidth
  - Reliability analysis on combined locality
- **ECWide:** design of a wide-stripe erasure-coded system
  - Combined locality for single-chunk repair and full-node repair
  - Efficient encoding via **multi-node encoding**
  - Efficient updates via **inner-rack parity updates**
  - Two ECWide prototypes: cold (ECWide-C) and hot (ECWide-H) storage
- Evaluation: single-chunk repair time reduced by 90.5% with ultra-low storage (1.063×)

Combined Locality

- Definition: (n,k,r,z) CL
  - (n,k,r) LRC + (n,k,z) TL
  - **c:** number of chunks of a stripe in a rack
  - **f:** number of tolerable node failures of a stripe
  - Requirement: $c \le f$; otherwise, a rack failure leads to data loss
- Design idea:
  - If c increases, a local inner-rack repair covers more chunks
    → reducing more cross-rack repair bandwidth
  - Minimum cross-rack repair bandwidth: **when c = f**
  - Selection of LRC: **Azure-LRC** has largest f under same (n,k,r)

Construction of CL: Azure-LRC coded chunks placed in racks satisfying c = f

Trade-off Analysis



LRC: (n,k,r) Azure-LRC

TL: (n,k,z) Topology Locality

CL: (n,k,r,z) Combined Locality

CL outperforms TL and LRC in terms of trade-off of redundancy and cross-rack repair bandwidth

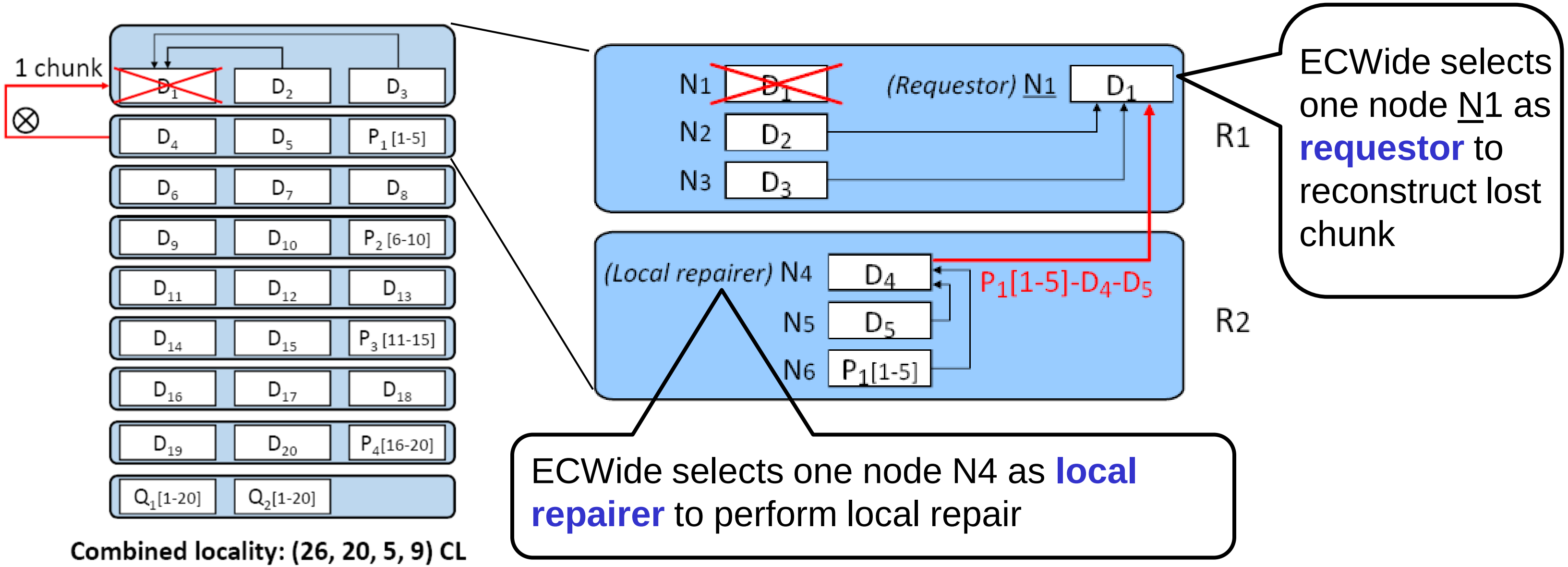
ECWide

- **ECWide**: a wide-stripe erasure-coded storage system

- Goals:
  - Minimum cross-rack repair bandwidth: realizes combined locality
  - Efficient encoding: proposes a multi-node encoding design
  - Efficient parity updates: proposes an inner-rack parity update design

Repair in ECWide

➢ Single-chunk repair:



Repair in ECWide

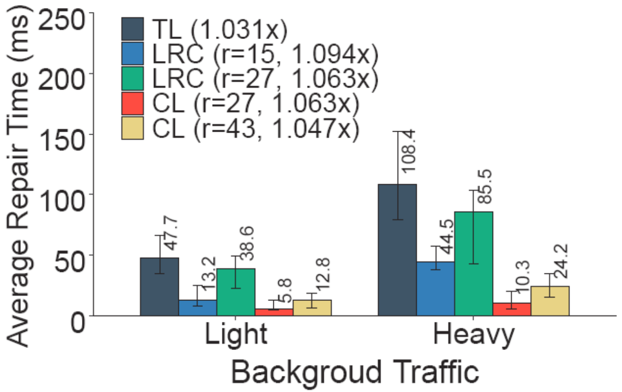
- Full-node repair:
  - Multiple single-chunk repairs in parallel
  - Problem: Different single-chunk repairs may choose identical nodes as requestors or local repairers → degraded parallel performance
  - Method: Always select least-recently-selected (LRS) nodes as requestors or local repairers
    - A doubly-linked list tracks which node has been recently selected
    - A hashmap holds the node ID and the node address of the list

Implementation

- Two ECWide prototypes:
  - **ECWide-C**: for cold storage
    - Large-sized chunks (e.g., 64MiB in HDFS)
    - Mainly implemented in Java with about 1,500 SLoC
    - Encoding implemented in C++ with about 300 SLoC on Intel ISA-L
  - **ECWide-H**: for hot storage
    - Small-size chunks (e.g. 4KiB [Zhang et al., FAST'16])
    - Built on Memcached
    - Extending libMemcached with about 3000 SLoC in C

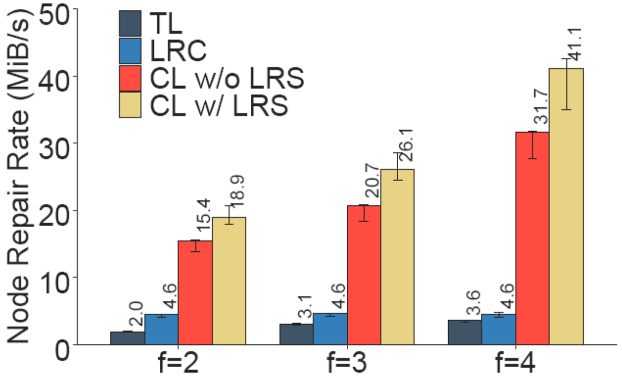
ECWide-H Experiments

- CL shows lower single-chunk repair time than TL (up to 90.5%) and LRC (up to 87.9%) with ultra-low redundancy (1.063)
- CL shows highest full-node repair rate; higher gain via LRS



$k = 128,\ f = 4$

Single-chunk repair



full-node repair

More experiments on ECWide-C and ECWide-H in the paper

Conclusions

- Propose combined locality to first address the wide-stripe repair problem systematically
- Design ECWide, a system that realizes combined locality, multi-node encoding, and inner-rack parity updates
- Implement ECWide for both cold and hot storage systems
- Show ECWide's efficiency in repair, encoding, and updates

ECWide source code: **https://github.com/yuchonghu/ecwide**

THANK YOU

Contacts:

Yuchong Hu yuchonghu@hust.edu.cn

Patrick Lee pclee@cse.cuhk.edu.hk