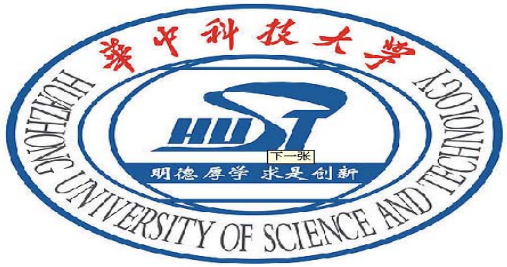
FastCDC: a Fast and Efficient Content-Defined Chunking Approach for Data Deduplication

Wen Xia,

Yukun Zhou, Hong Jiang, Dan Feng, Yu Hua,

Yuchong Hu, Yucheng Zhang, Qing Liu



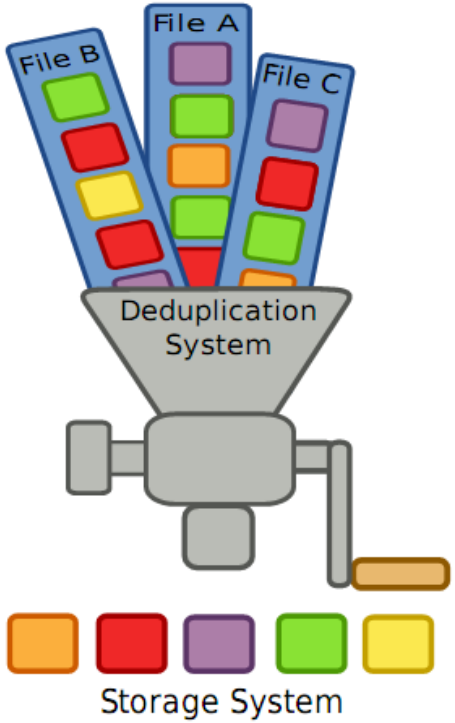


SANGFOR



Outline

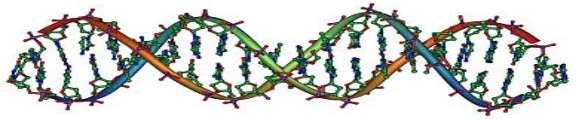
- Background and Motivation
- Design and Implementation
- Performance Evaluation
- Conclusion



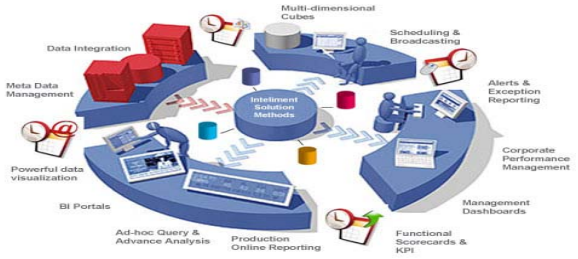
Redundancy Data in the Big Data Era

- Explosive growth of digital data
  - 2007，281 EB
  - 2010，1.2 ZB
  - 2011，1.8 ZB
  - 2020，44 ZB
  - GB,TB, PB, EB, ZB, YB, …
- **Data Reduction for Backup**
  - Unstructured and Cumbersome
  - Duplicate or Redundant



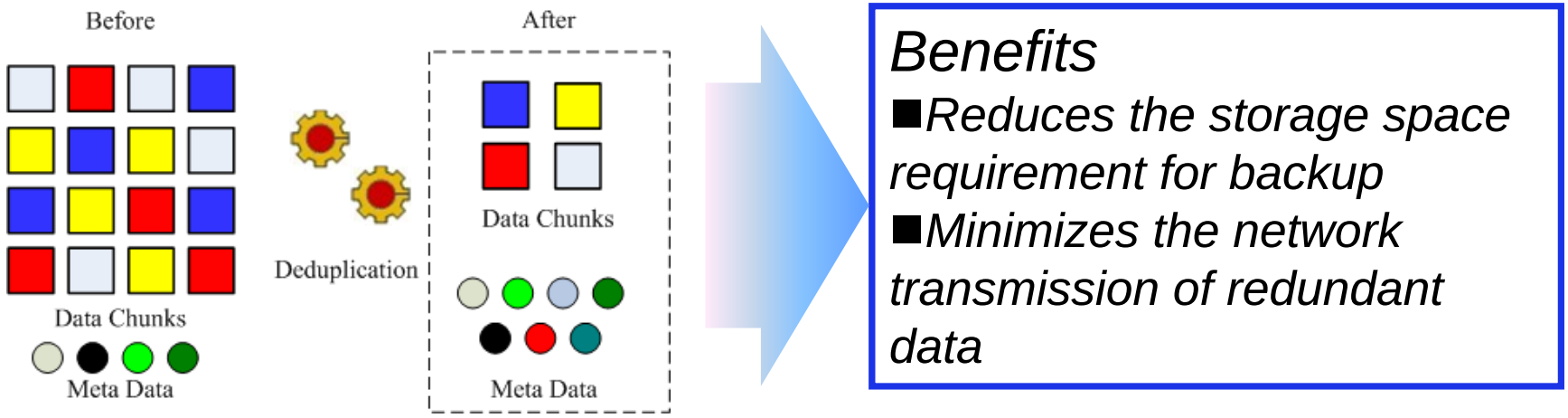






Data Deduplication

- A global compression approach for storage systems
  - Identifying the duplicate chunks (size of about 8KB)
  - Using a hash signature to uniquely identify a data chunk
  - Secure hash signature: MD5, SHA-1, SHA-256, …

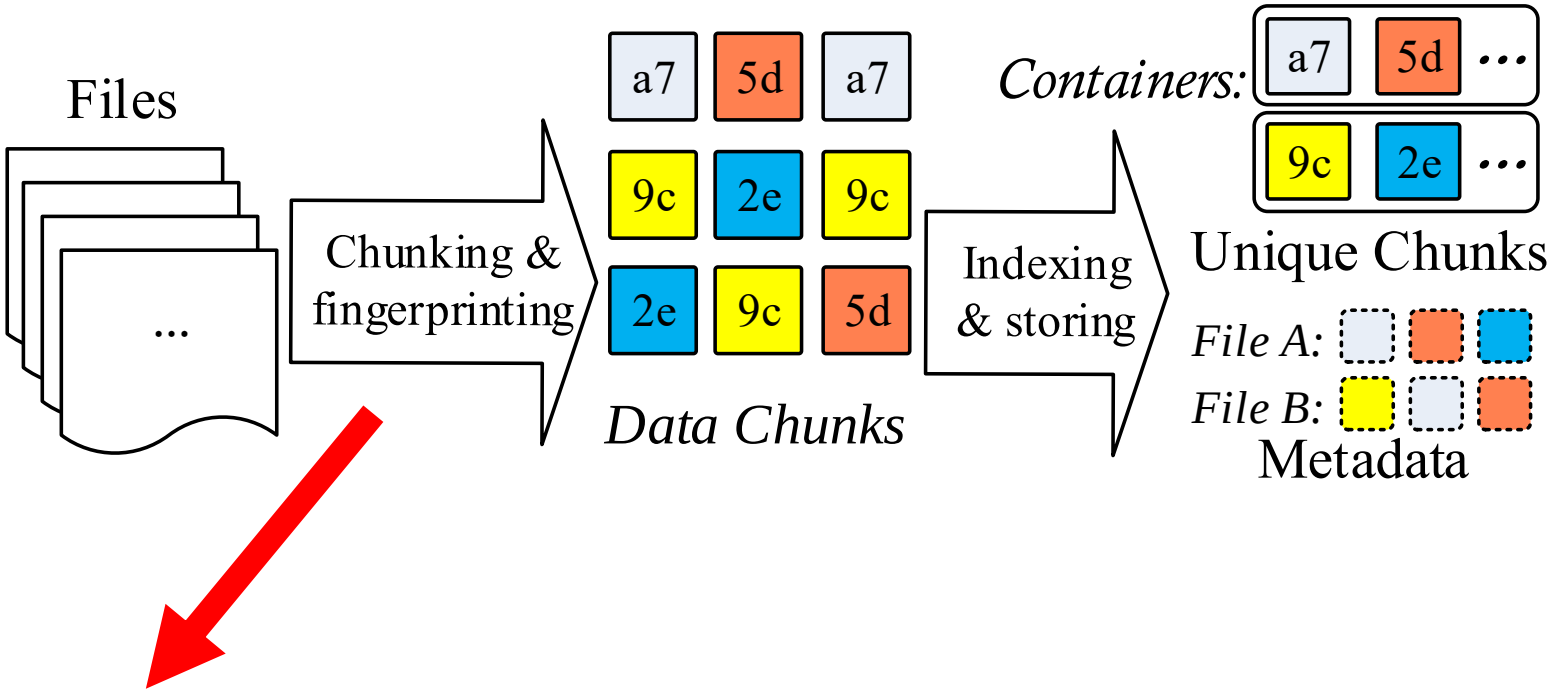


Benefits

■*Reduces the storage space requirement for backup*

■*Minimizes the network transmission of redundant data*

Deduplication Workflow



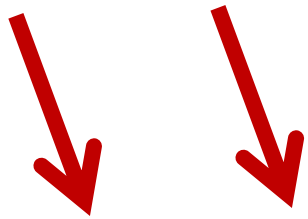
- We focus designing a fast and efficient chunking approach in this paper

Content-Defined Chunking (CDC)

- CDC declares the chunk boundaries based on the contents instead of on the byte offset
- It resolves the **boundary-shift problem** facing Fixed-Size Chunking (FSC)

File A

5865 3589 7532 3946 2643 3832 7950 2889 …

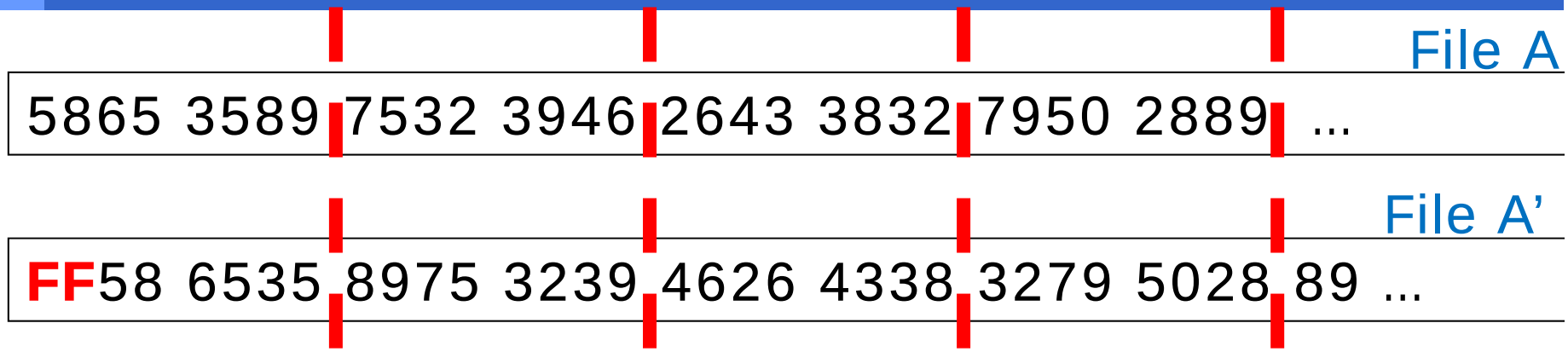


boundary-shift …

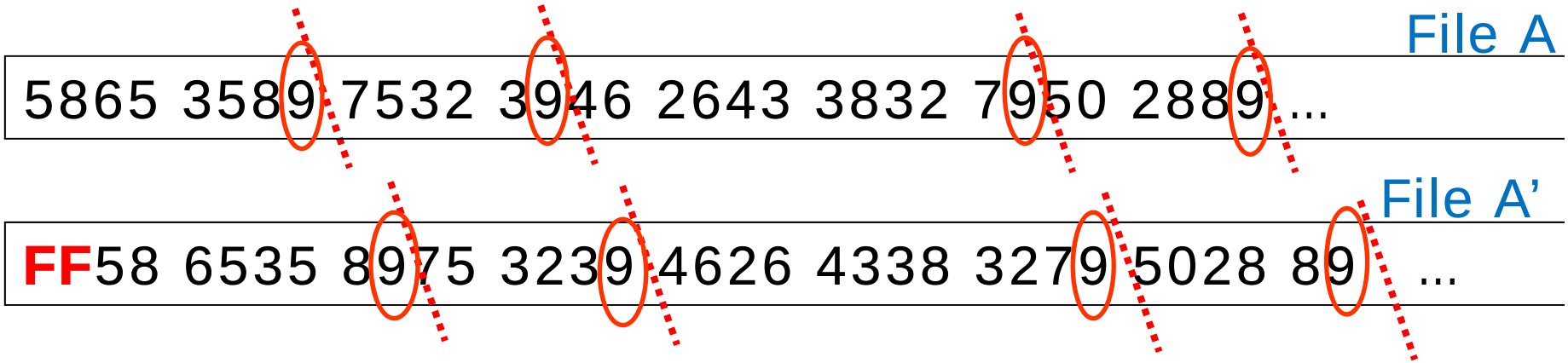
File A'

FF58 6535 8975 3239 4626 4338 3279 5028 89 …

FSC vs. CDC



➔➔*FSC: No duplicates will be detected*



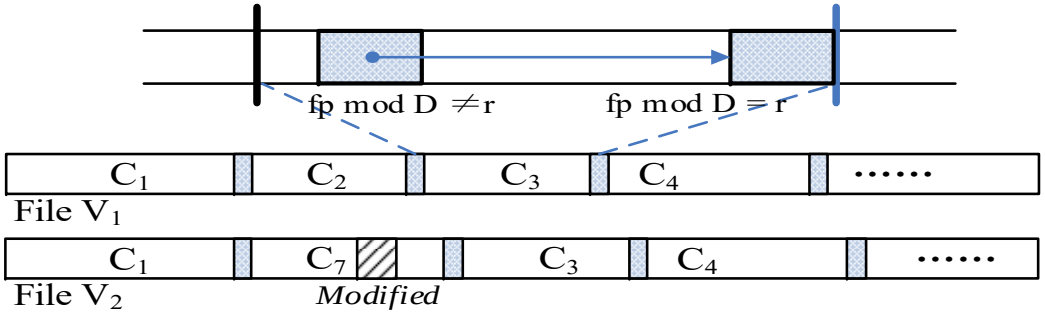
➔➔*CDC: Most duplicates will be detected*

The Classic Rabin-based CDC

- LBFS employs Rabin hash for CDC

$$Rabin(B_1, B_2, ..., B_\alpha) = A(p) = \{\sum_{x=1}^{\alpha} B_x p^{\alpha - x}\} mod\ D \quad (1)$$

- It uses a sliding-window on the contents and computes Rabin hash of the window



  - Cut a chunk if *fp mod D == r*
  - Otherwise continue sliding to find cut-point …

The Classic Rabin-based CDC

- Rabin is a rolling hash since it is able to be computed from the previous value
  - Fast and efficient for CDC process
- Influence on chunk size
  - Variable size (due to the randomness of Rabin hash)
  - Expected length is approx. N (e.g., 8KB)
- Define Min/Max Chunk Size
  - To avoid chunks of extremely large or small sizes
  - Typically 2KB and 64KB when avg. 8KB

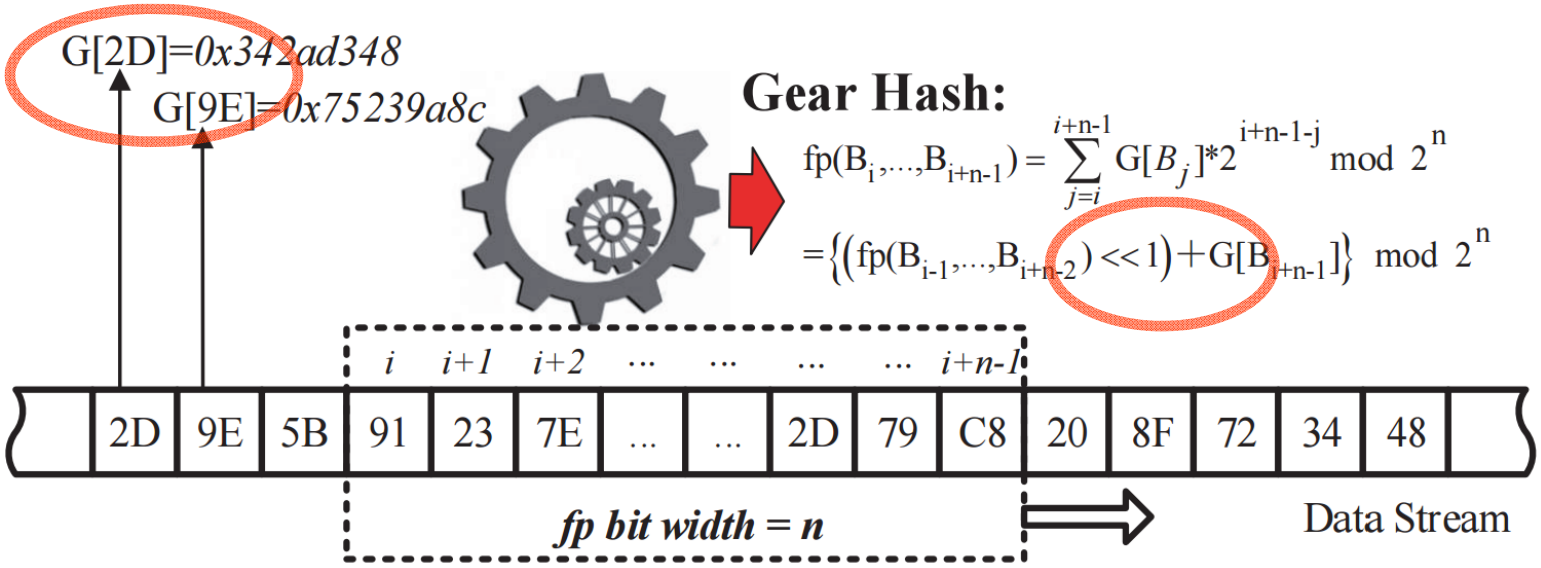
State of Art

- Challenges: still time-consuming
  - Compute and judge the hashes byte-by-byte
  - A potential bottleneck in dedup systems

- Algorithmic-oriented CDC optimizations
  - *SampleByte (NSDI'10), Gear (Performance'14), AE (Infocom'15), etc.*
- Hardware-oriented CDC optimizations
  - *GPGPU, multicore/manycore*
  - *StoreGPU (HPDC'08), Shredder (FAST'12), P-Dedupe (NAS'12), etc.*

Gear-based CDC

- Gear: a very fast rolling hash for CDC



Implementation: *fp = (fp << 1) + G(b)*

- One array lookup
- One "+" operation
- One left-shift operation

Ddelta: A dedup-inspired fast delta compression approach @ Performance'14

Motivations

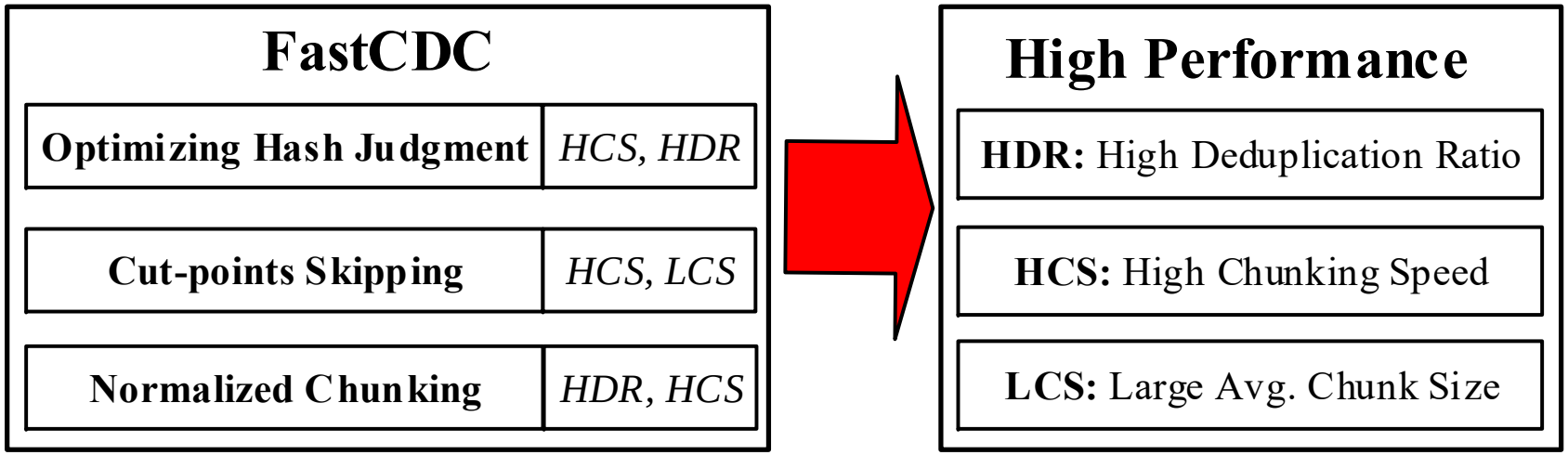
- Can Gear-based CDC be faster?

1. For the two stages of CDC, hashing is fast enough, how about the stage of hash judgment?

2. As most of chunk sizes are approximately 8KB, can we skip some regions before chunking to reduce some unnecessary computations?

FastCDC Overview

- Improving chunking performance over Gear-based CDC



Optimizing Hash Judgment

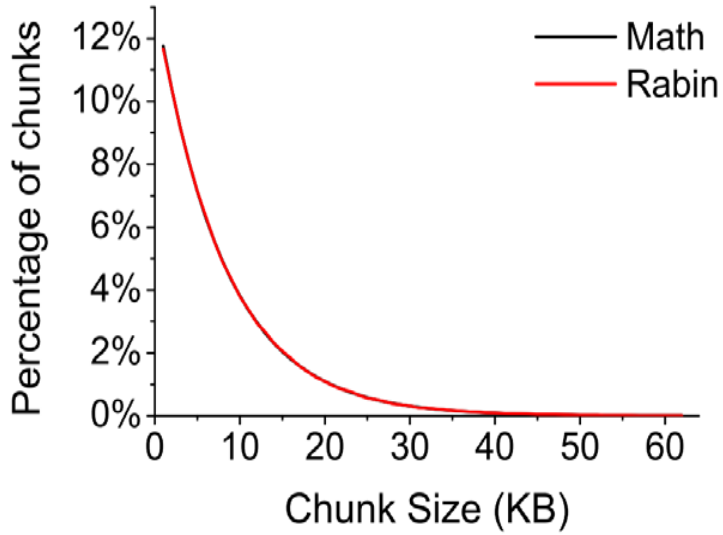
- **Zero-padding, to enlarge the sliding window**
  - Gear-based chunking: *fp & (2^n-1) == r*
  - Small sliding window size results in low dedup ratio
  - Padding several zero bits into the mask value (i.e., *2^n-1*) enlarges the effective masked bit-width of *fp*, → **The enlarged sliding window size improves the dedup ratio**
- **Simplified hash judgment speeds up CDC**
  - *fp mod D == r* → *! (fp & mask)*
  - *Speeds up the stage of hash judgment for CDC*

Cut-point Skipping

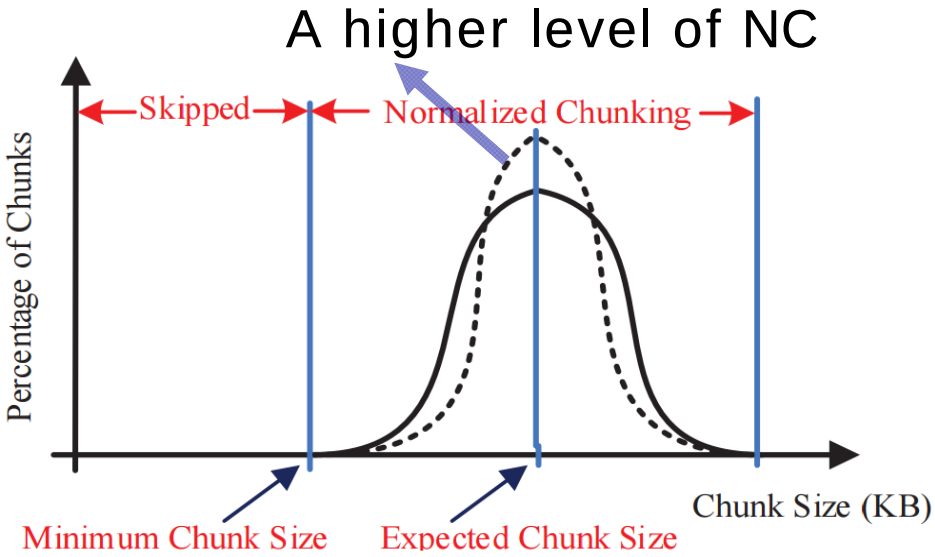
- The limit on minimum chunk size
  - Avoids generating extremely small-sized chunks
  - Also can be used for cut-point skipping, which reduces computation and thus further speeds up CDC
- **But …**
  - It results in some chunks not truly divided by contents but forced by the cut-point skipping
  - According to our study, the chunks smaller than 2KB account for about 22.1% of the total
  - Thus this approach will decrease dedup ratio

Normalized Chunking (NC)

- A novel chunking approach
  - To solve the problem of decreased dedupication ratio facing **"cut-point skipping"**
  - To generate chunks of sizes normalized to a specified region that is always larger than the min chunk size



exponential distribution

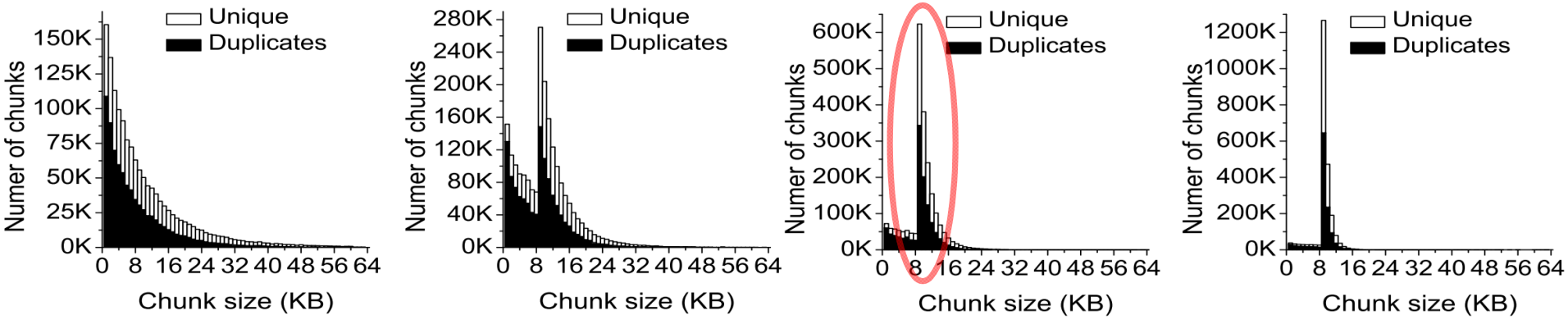


normal distribution

Implementation of Normalized Chunking (NC)

- For the traditional CDC (avg. 8KB chunk size)
  - *fp & 0x1fff==r, 13* effective mask bits
- For the normalized chunking
  - When the offset is smaller than 8KB
    - *E.g., fp & 0x7fff==r, 14 effective mask bits*
    - → harder to cut chunks (smaller than 8KB)
  - When the offset is larger than 8KB
    - *E.g., fp & 0x0fff==r, 12 effective mask bits*
    - → easier to cut chunks (larger than 8KB)

Chunk-Size Distribution of NC



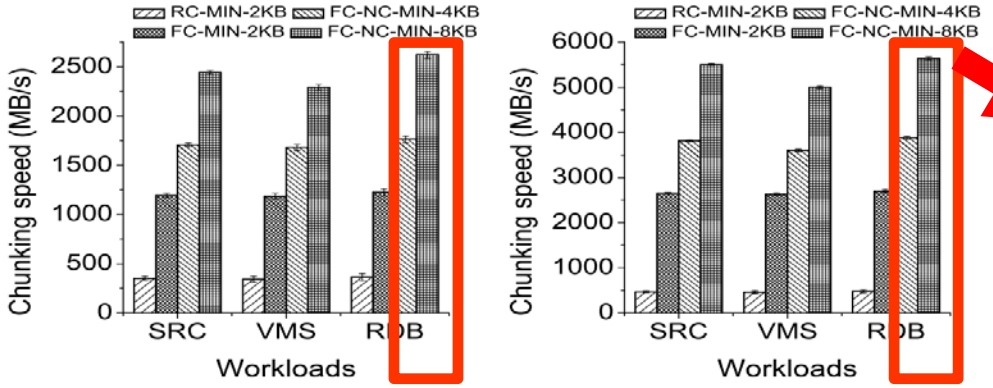
(a) FastCDC without NC (b) FastCDC with NC level 1 (c) FastCDC with NC level 2 (d) FastCDC with NC level 3

- NC generates chunks approximately normalized to 8KB
  - Makes it possible to combine with cut-point skipping
  - Improves dedup ratio by reducing the larger chunks
  - No additional computing and comparing operations

Performance Evaluation

- Experimental platform
  - Runs FastCDC on Ubuntu 12.04.2 operating system
  - Intel i7-4770 (@3.4GHz) and i7-930 (@2.8GHz) processors
- Seven workloads
  - Source code files, backup tarred files, VM images, database files, etc.
  - Total size of about 5TB
- Three key metrics
  - Chunking speed (in memory)
  - Deduplication ratio
  - Average chunk size (after chunking)

Putting It All Together (Chunking Speed)



(a) Speed on intel i7-930 (b) Speed on intel i7-4770

Figure 13: Chunking speed of the four CDC approaches.

FastCDC is about 10X faster than Rabin-based CDC

- Using Gear for Hashing
- Optimizing hash judgment
- Cut-point skipping
- Normalizing chunking

Approaches	Instructions	IPC	CPU cycles
RC-MIN-2KB	38, 829, 037	2.35	16, 537, 973
FC-MIN-2KB	15, 074, 950	4.37	3, 452, 146
FC-NC-MIN-4KB	11, 008, 372	4.82	2, 284, 453
FC-NC-MIN-8KB	7, 750, 124	4.82	1, 608, 033

Table 6: Number of instructions, instructions per cycle (IPC), and CPU cycles required to chunk 1MB data by the four CDC approaches on the Intel i7-4770 processor.

FastCDC uses far fewer instructions and CPU cycles

Put It All Together (Dedup ratio)

Rabin-based CDC **FastCDC with diff. min chunk size**

Dataset	RC w/ Min2KB	FC w/ Min2KB	FC-NC w/ Min4KB	FC-NC w/ Min8KB	XC w/ 10KB
TAR	47.58%	47.64%	50.19%	47.18 %	12.21%
LNX	97.25%	97.26%	97.35%	97.10%	96.51%
WEB	95.09%	94.02%	95.47%	94.44%	93.19%
VMA	38.23%	37.96%	40.31%	38.15%	18.26%
VMB	96.13%	96.09%	96.24%	96.11%	95.68%
RDB	95.53%	95.50%	96.71%	95.70%	9.80%
SYN	93.64%	93.67%	94.09%	92.62%	75.06%

- FastCDC achieves nearly the same (Min8KB) or higher (Min4KB) dedup ratio than Rabin-based CDC

Put It All Together (Avg. chunk size)

Rabin-based CDC **FastCDC with diff. min chunk size**

Dataset	RC w/ Min2KB	FC w/ Min2KB	FC-NC w/ Min4KB	FC-NC w/ Min8KB	XC w/ 10KB
TAR	11873	12192	10347	14076	10240
LNX	5978	5969	6288	7585	6477
WEB	9985	10725	9327	12862	9513
VMA	12743	12787	11161	15031	10239
VMB	11937	12138	10850	15148	10239
RDB	10232	10238	9751	13846	10240
SYN	10954	10945	10318	14123	10240

- FastCDC generates nearly the same (Min4KB) or much larger (Min8KB) average chunk size, which means less metadata overheads for dedup indexing

Conclusion

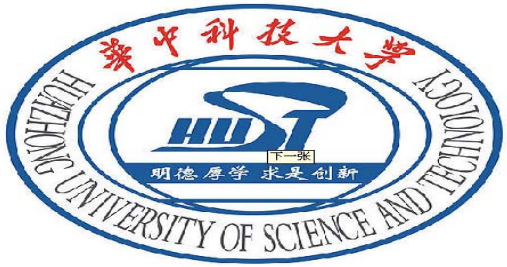
- **FastCDC: improving CDC performance**
  - Optimizing the hash judgment
  - Cut-point skipping
  - Normalized chunking
- FastCDC is shown to be 10X faster than Rabin-based CDC and 3X faster than Gear-based CDC while achieving a comparable dedup ratio

- In our future work:
  - Incorporate FastCDC in some other deduplication systems
  - Release the FastCDC source code to be shared with the storage systems research community

Thank You !

Questions?

I can be reach at: xia@hust.edu.cn
Or wxia.hustbackup.cn





SANGFOR



UNIVERSITY OF
TEXAS
ARLINGTON