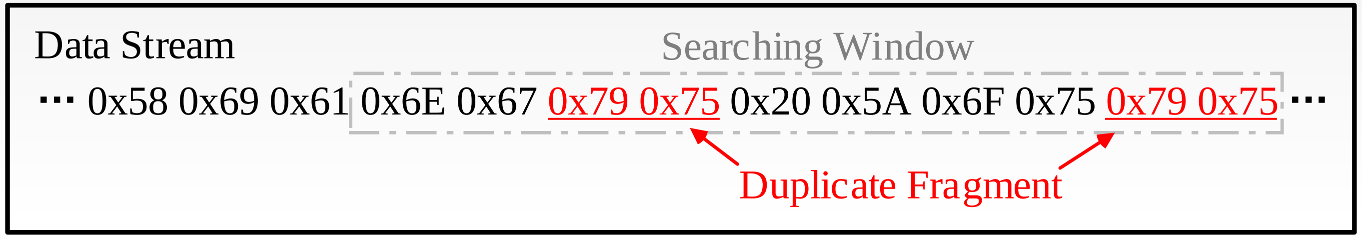
Building a High-performance Fine-grained Deduplication Framework for Backup Storage with High Deduplication Ratio

Xiangyu Zou[1], Wen Xia[1], Philip Shilane[2], Haijun Zhang[1], and Xuan Wang[1]

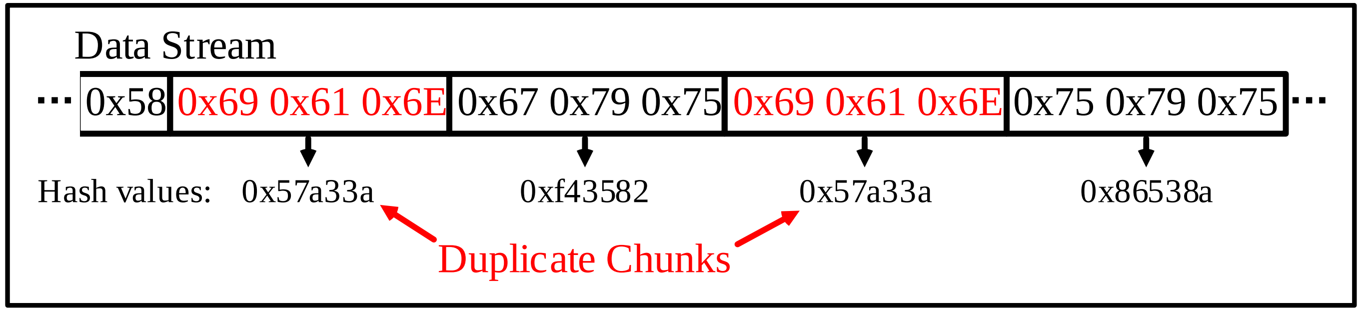
[1]Harbin Institute of Technology, Shenzhen; [2]Dell Technologies

Data Reduction

- How to reduce storage cost
  - General Compression
    - For usual-size files
    - String-level
    - Limited window



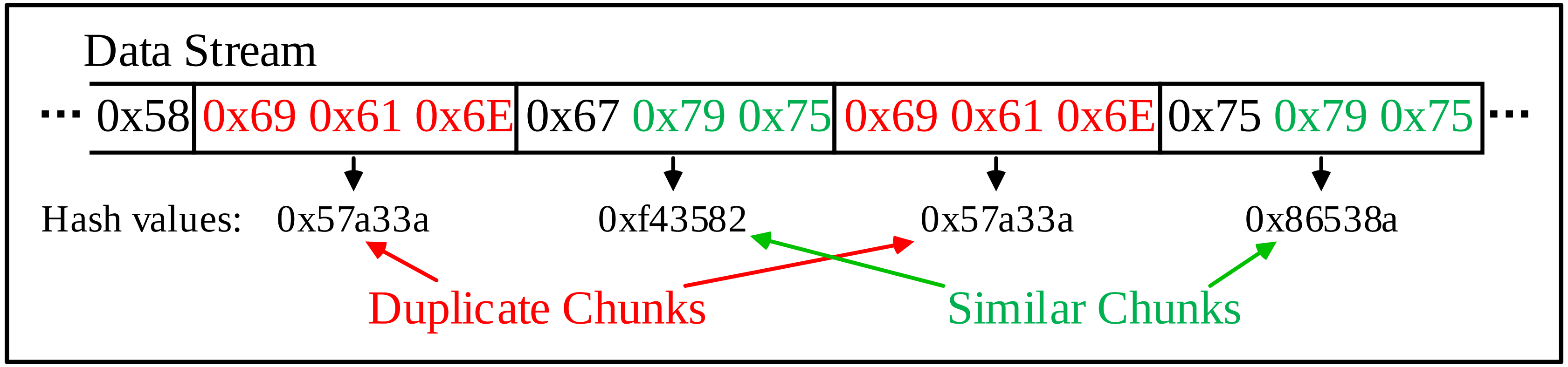
  - Deduplication
    - For very large files
    - Chunk-level
    - Global

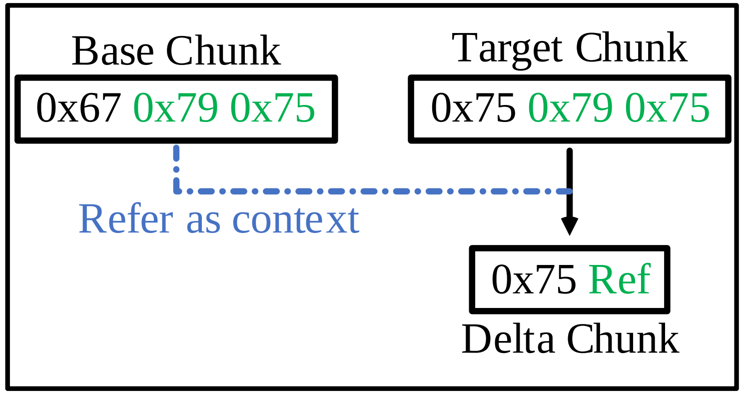


  - Both have been widely used in storage products
  - Can not fully utilize data’s compressibility

Fine-Grained deduplication

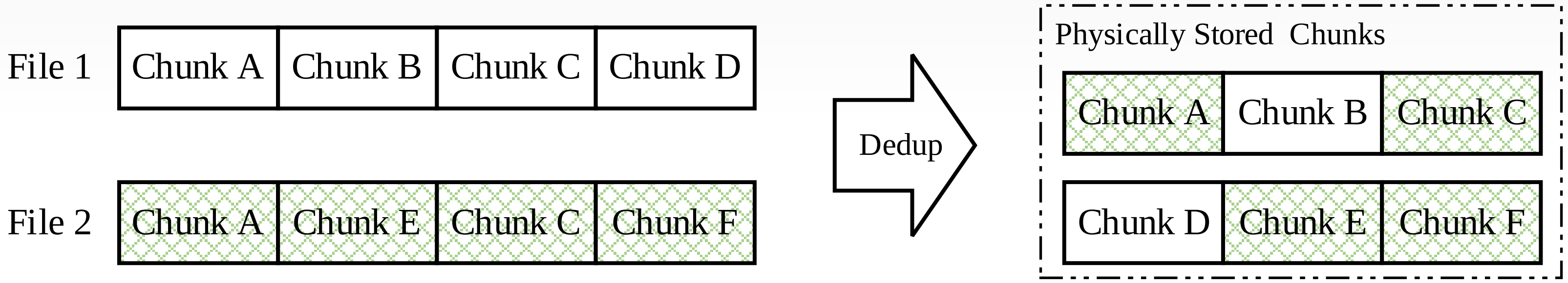
- Fine-grained deduplication
  - Leverages not only identical chunks, but also similar chunks
  - Introduces additional steps on post-deduplication chunks
    - Detects similar chunks for an unduplicated chunk (i.e., target chunk for delta encoding)
    - Reads back a similar one (an already stored chunk) as a base chunk
    - Calculates delta difference between the target chunk and the base chunk
  - String-level, Global





However... What is the cost?

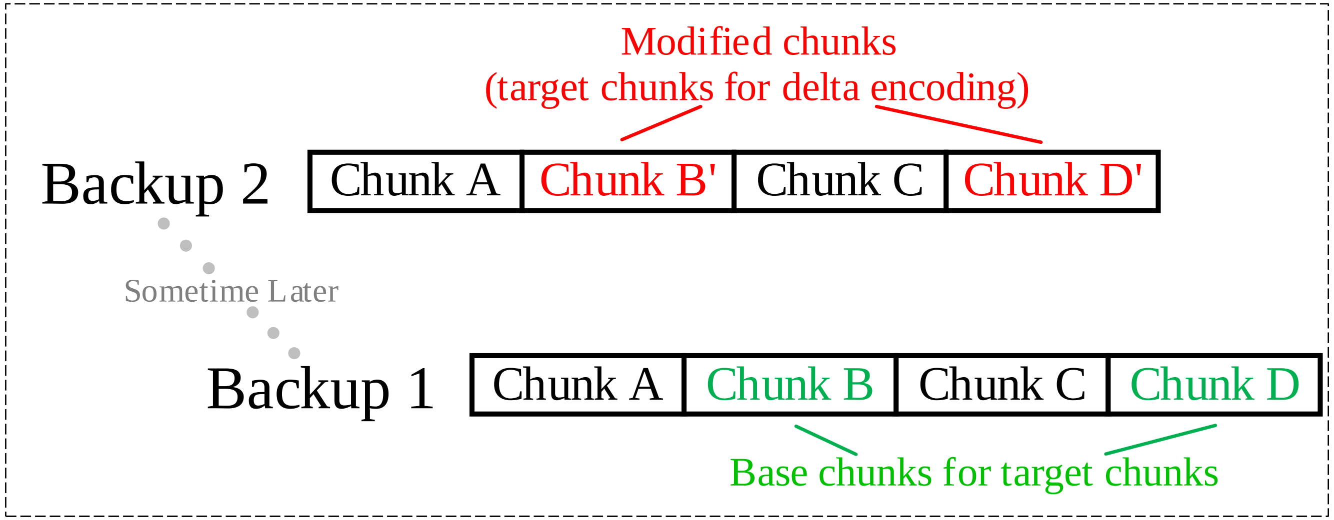
- Reusing data hurts locality
  - Declines systems' performance

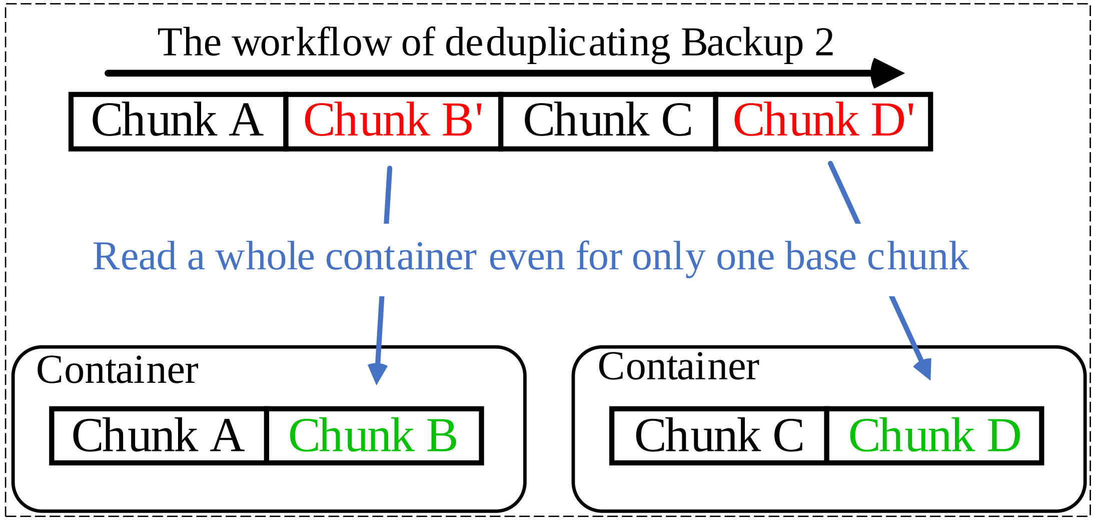


- Fine-grained deduplication introduces a new form of data reuse
  - Additional locality issues
  - Gates performance of both the deduplication and the restore workflows (i.e., write & read path)
  - This work aims to address these issues

Additional locality issues #1
- Poor locality in base chunks (the write path)

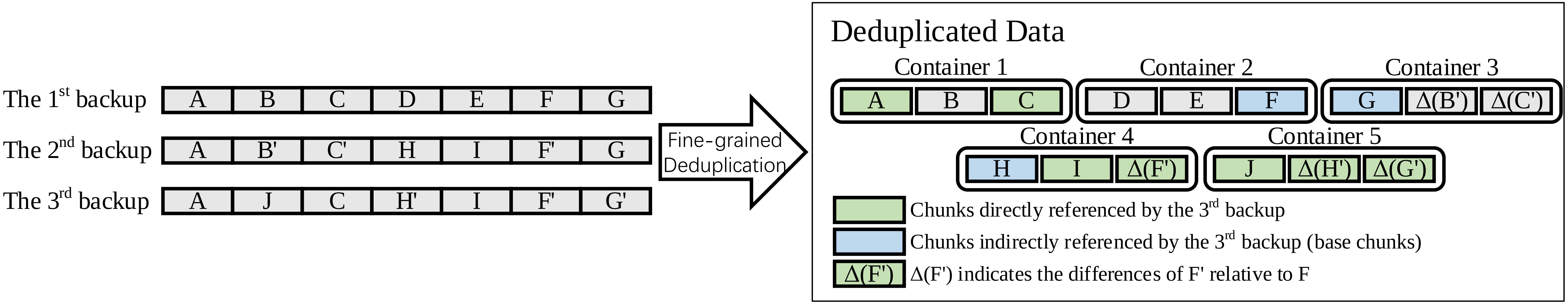
- Causes:
  - The distribution of base chunks' physical positions is random
  - Consecutive chunks are usually compressed together (local compression)
    - Accessing the whole compression unit (e.g., container) even for only one chunk
- Results:
  - Need to read a whole container even for only one base chunk
  - Inefficient I/O when reading base chunks in the write path





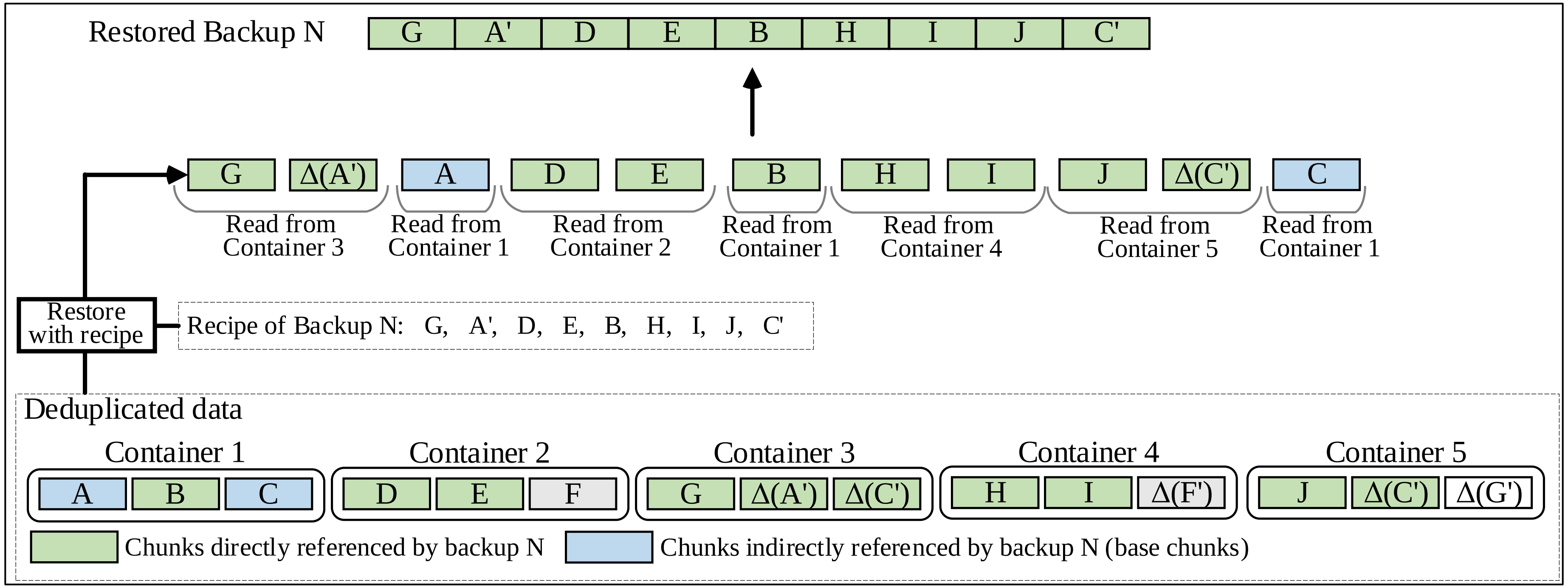
Additional locality issues #2
- Poor locality in restore-required chunks (the read path)

- Causes:
  - Two kinds of reference relationships
    - Backup workloads – Chunks (introduced by chunk-level deduplication)
    - Delta chunks – Base chunks (additionally introduced by delta encoding)
    - Aggravate the fragmentation problem
  - Local compression leads to a large I/O unit
- Results:
  - Inefficient I/O when reading restore-required chunks in the read path



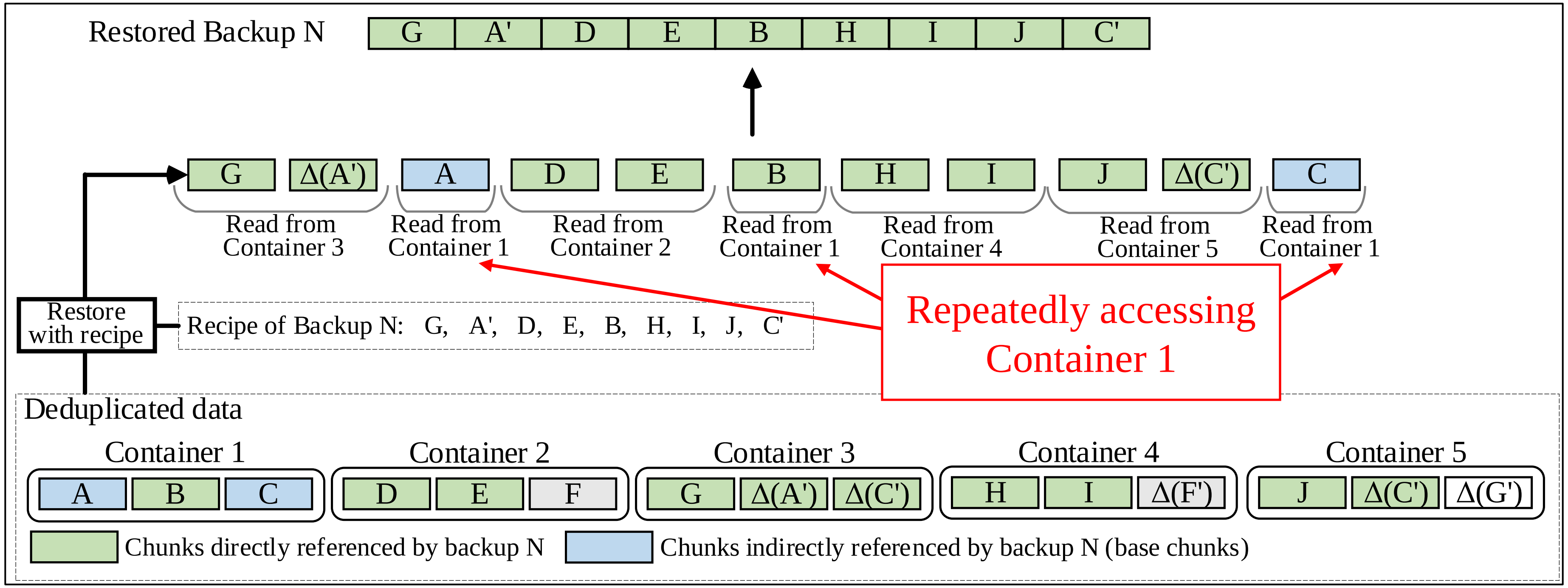
Additional locality issues #3
- Poor locality in delta-base pairs (the read path)

- Causes:
  - Traversing restore-required chunks when restoring a deduplicated backup
  - Delta chunks have dependencies, but usually are far away from their bases
- Results:
  - Repeatedly accessing containers in the read path



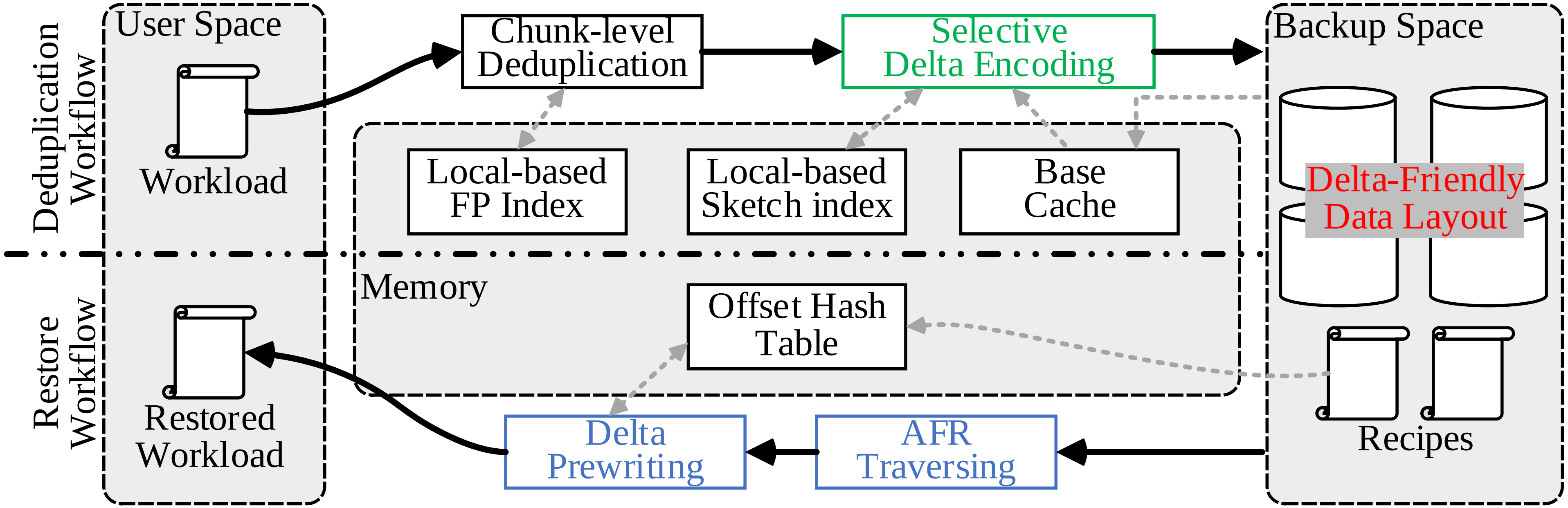
Additional locality issues #3
- Poor locality in delta-base pairs (the read path)

- Causes:
  - Traversing restore-required chunks when restoring a deduplicated backup
  - Delta chunks have dependencies, but usually are far away from their bases
- Results:
  - Repeatedly accessing containers in the read path



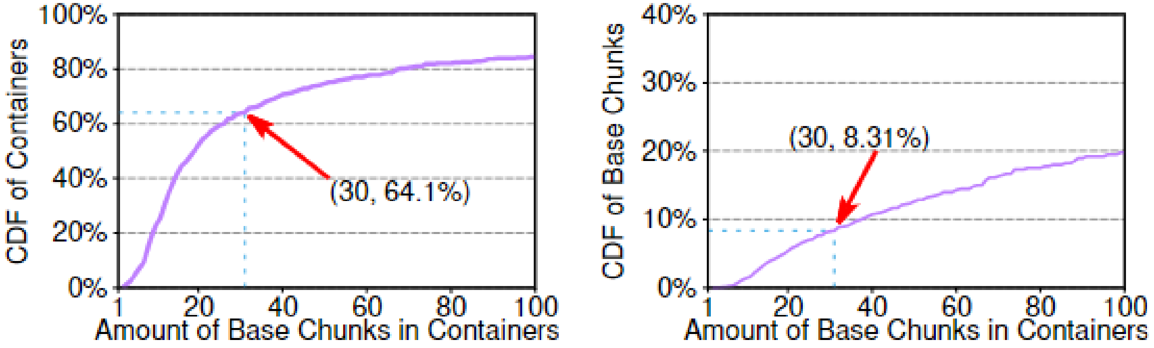
Solution

- Techniques to address these three additional locality issues
  - Selective Delta Encoding
  - Delta-friendly Data Layout
  - Always-Forward-Reference Traversing and Delta Prewriting
- A fine-grained deduplication framework – MeGA



Selective Delta Encoding

- An observation: Base chunks are not distributed evenly
- For example, in an evaluated dataset:
  - 64.1% containers hold ~30 base chunks (“base-sparse containers”)
  - These 64.1% containers only includes 8.31% of the total base chunks.
- Skip delta encoding if base chunks are in base-sparse containers
- Avoids reading these “inefficient” containers in the deduplication workflow

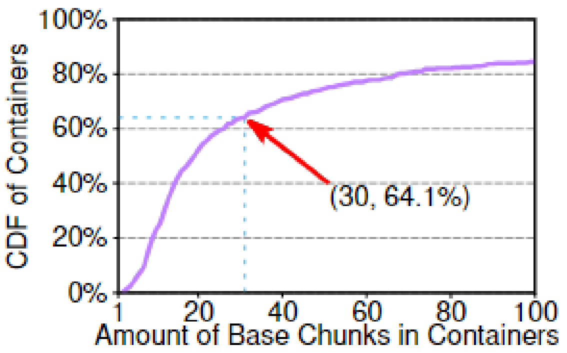


(a) 64.1% of containers contain only ~30 base chunks.

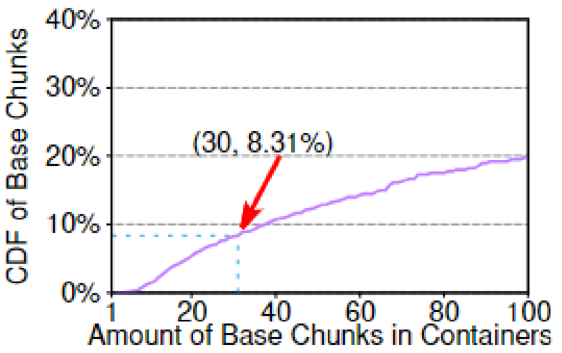
(b) These 64.1% containers only includes 8.31% of the total base chunks.

Selective Delta Encoding

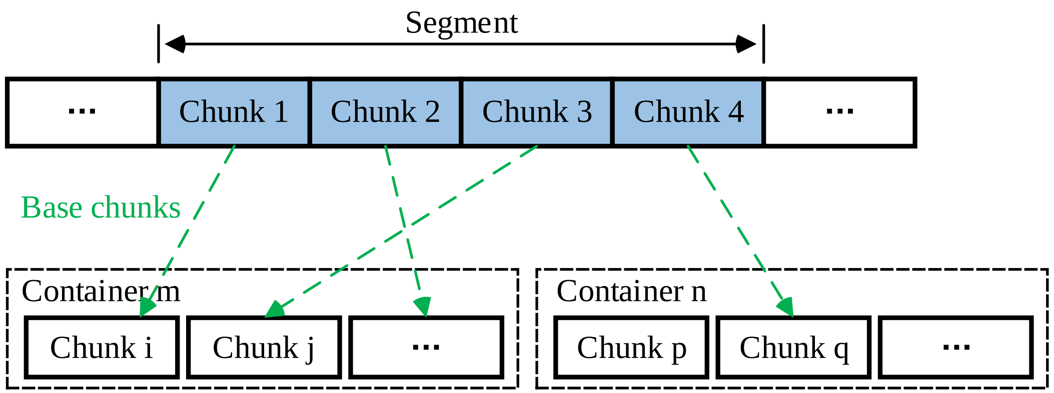
- An observation: Base chunks are not distributed evenly
- For example, in an evaluated dataset:
  - 64.1% containers hold ~30 base chunks (“base-sparse containers”)
  - These 64.1% containers only includes 8.31% of the total base chunks.
- Skip delta encoding if base chunks are in base-sparse containers
- Avoids reading these “inefficient” containers in the deduplication workflow



(a) 64.1% of containers contain only ~30 base chunks.

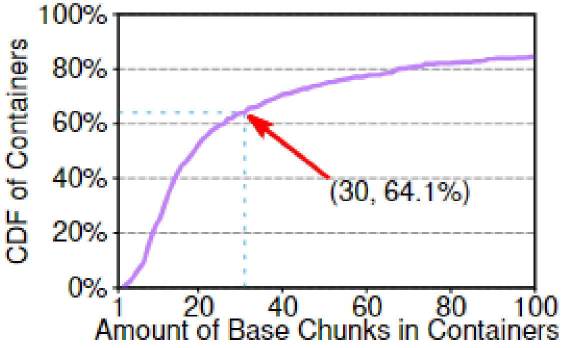


(b) These 64.1% containers only includes 8.31% of the total base chunks.

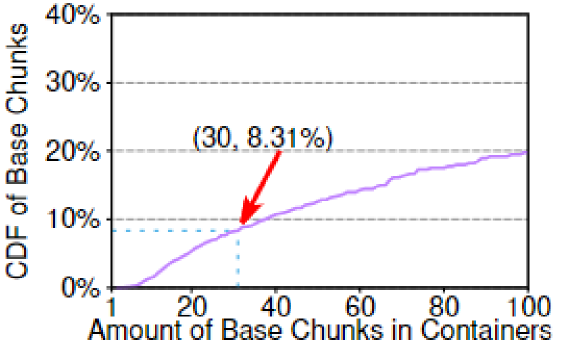


Selective Delta Encoding

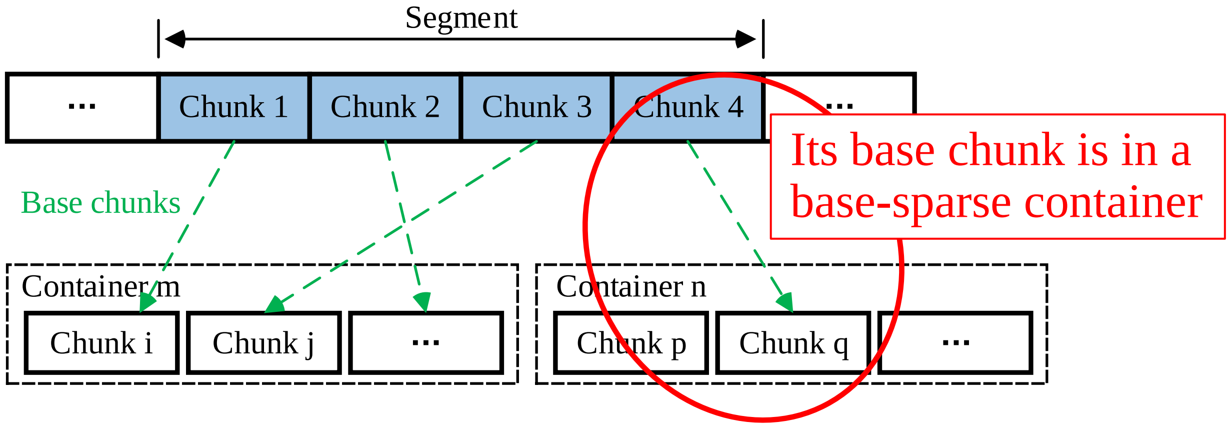
- An observation: Base chunks are not distributed evenly
- For example, in an evaluated dataset:
  - 64.1% containers hold ~30 base chunks (“base-sparse containers”)
  - These 64.1% containers only includes 8.31% of the total base chunks.
- Skip delta encoding if base chunks are in base-sparse containers
- Avoids reading these “inefficient” containers in the deduplication workflow



(a) 64.1% of containers contain only ~30 base chunks.

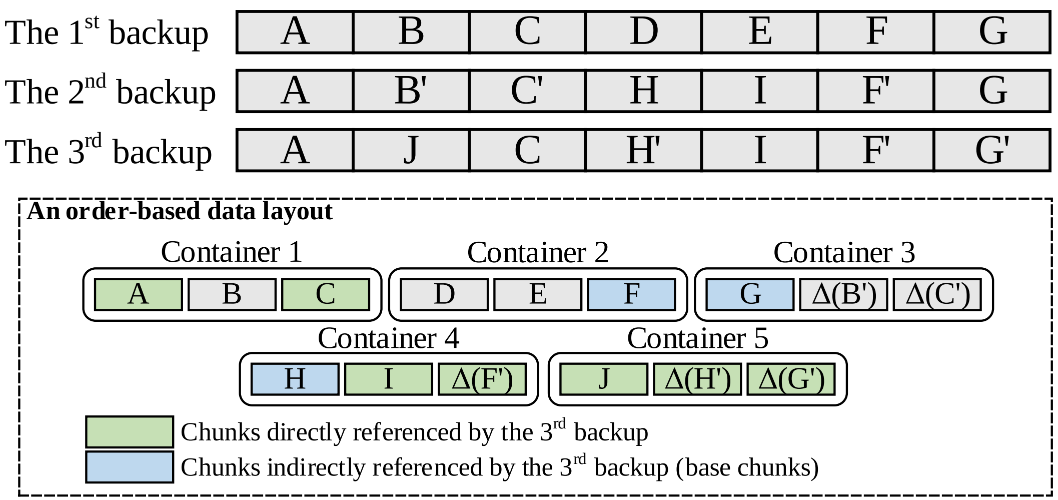


(b) These 64.1% containers only includes 8.31% of the total base chunks.



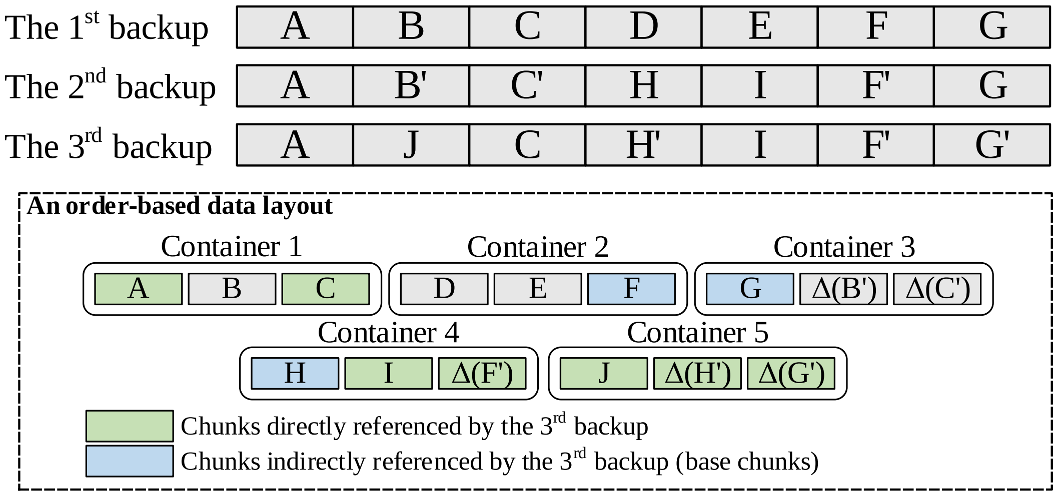
Delta-friendly Data Layout

- Consider two kinds of reference relationship
  - The “Necessary Chunks” of a backup
    - The combination of a backup’s directly and indirectly referenced chunks
  - The lifecycle of a chunk
    - A set of backups whose "Necessary chunks" includes this chunks.
- Lifecycle-based classification
- Avoids reading sparse containers in the restore workflow



Delta-friendly Data Layout

- Consider two kinds of reference relationship
  - The “Necessary Chunks” of a backup
    - The combination of a backup’s directly and indirectly referenced chunks
  - The lifecycle of a chunk
    - A set of backups whose "Necessary chunks" includes this chunks.
- Lifecycle-based classification
- Avoids reading sparse containers in the restore workflow

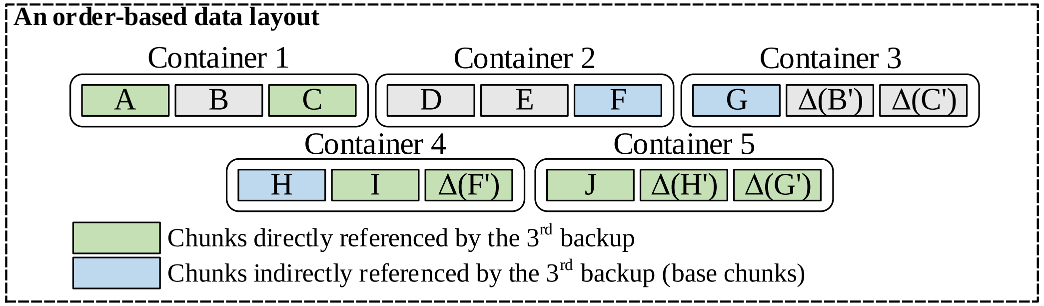


- NC_Backup1: A, B, C, D, E, F, G
- NC_Backup2: A, B, Δ(B’), C, Δ(C’), H, I, F, Δ(F’), G
- NC_Backup3: A, J, C, H, Δ(H’), I, F, Δ(F’), G, Δ(G’)

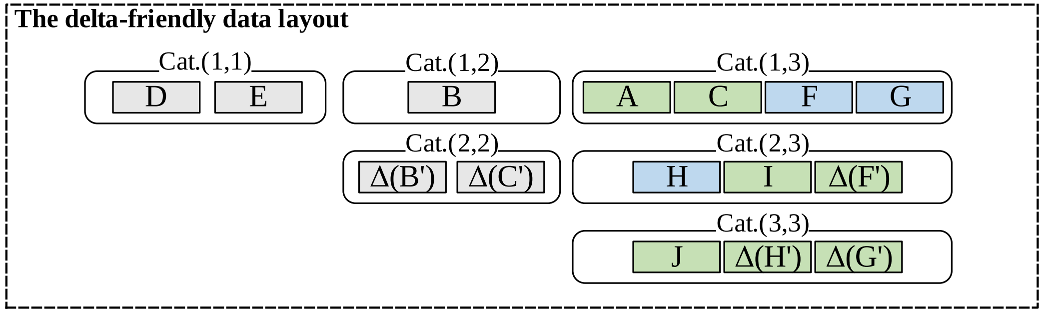
Delta-friendly Data Layout

- Consider two kinds of reference relationship
  - The “Necessary Chunks” of a backup
    - The combination of a backup’s directly and indirectly referenced chunks
  - The lifecycle of a chunk
    - A set of backups whose "Necessary chunks" includes this chunks.
- Lifecycle-based classification
- Avoids reading sparse containers in the restore workflow

The 1st backup	A	B	C	D	E	F	G
The 2nd backup	A	B'	C'	H	I	F'	G
The 3rd backup	A	J	C	H'	I	F'	G'

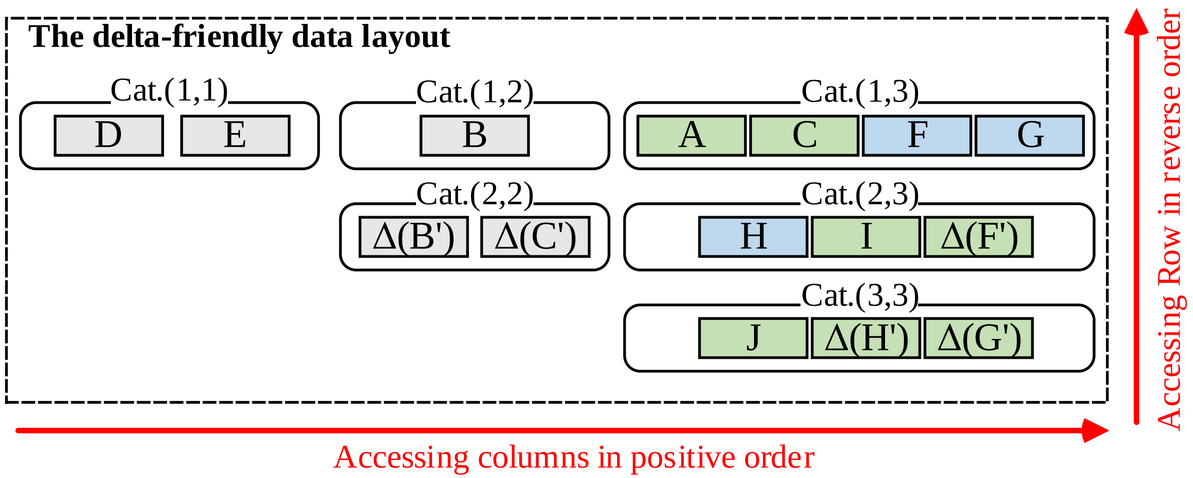


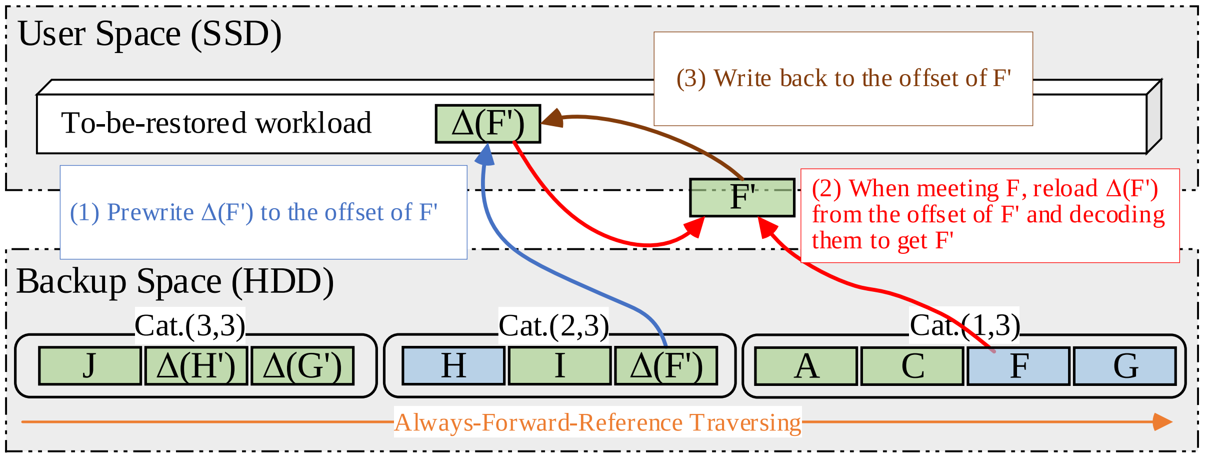
- NC_Backup1: A, B, C, D, E, F, G
- NC_Backup2: A, B, Δ(B’), C, Δ(C’), H, I, F, Δ(F’), G
- NC_Backup3: A, J, C, H, Δ(H’), I, F, Δ(F’), G, Δ(G’)



Always-Forward-Reference Traversing and Delta Prewriting

- A special path to traverse the restore-required chunks
  - Promises that delta chunks always appear before their base chunks
  - Rules to achieve AFR traversing
- Prewriting delta chunks
  - Asymmetric I/O characteristics of backup's/user's storage media
- Avoids repeatedly accessing restore-required chunks/containers





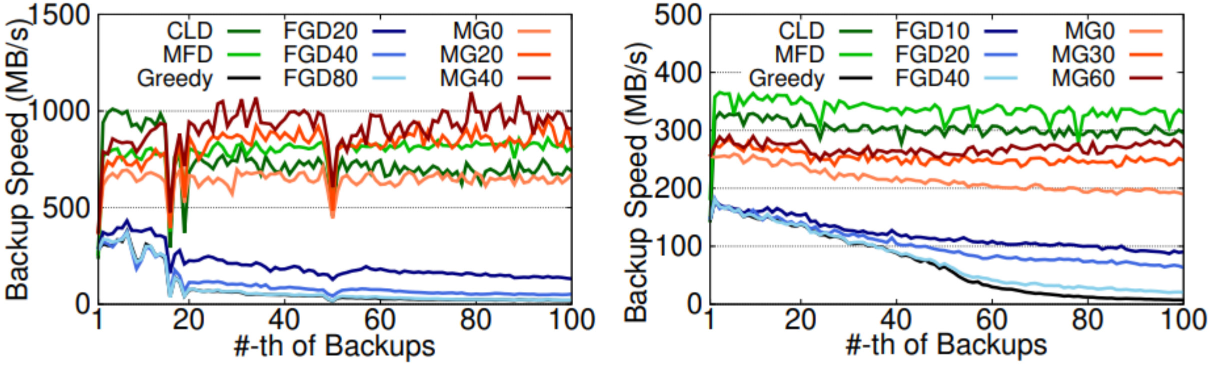
Evaluation

- Evaluated approaches
  - MeGA Our proposed approach, using the three proposed techniques
  - Greedy A fine-grained dedup approach with a greedy strategy
  - FGD A fine-grained dedup approach with the Capping rewriting technique
  - CLD A chunk-level dedup approach with Capping rewrite technique
  - MFD A chunk-level dedup approach with an optimized data layout
- Datasets

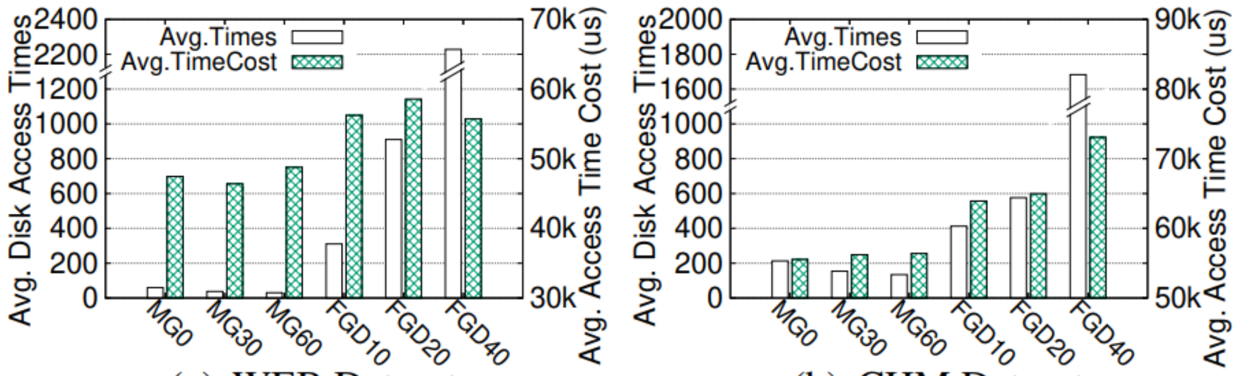
Name	Original Size	Versions	Workload Descriptions
WEB	269 GB	100	Backups of website: news.sina.com, captured from Jun. to Sep. in 2016
CHM	279 GB	100	Source codes of Chromium project from v82.0.4066 to v85.0.4165
SYN	1.38 TB	200	Synthetic backups by simulating file create/delete/modify operations
VMS	1.55 TB	100	Backups of an Ubuntu 12.04 Virtual Machine

Evaluations on the deduplication workflow

- Two parts:
  - The backup speed and statistics about accessing disks for reading bases
- Applying several parameters for FGD and MeGA
- MeGA achieves a 4.47–34.45× higher backup speed than Greedy
- Selective Delta Encoding hugely reduces disk accessing times
- Skipping more delta encoding will lead to a better speed.



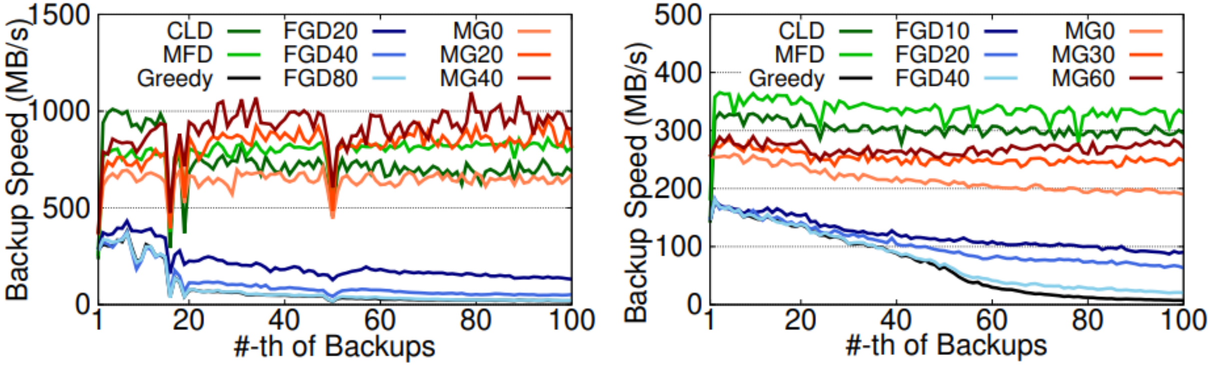
(a) WEB Dataset (b) CHM Dataset



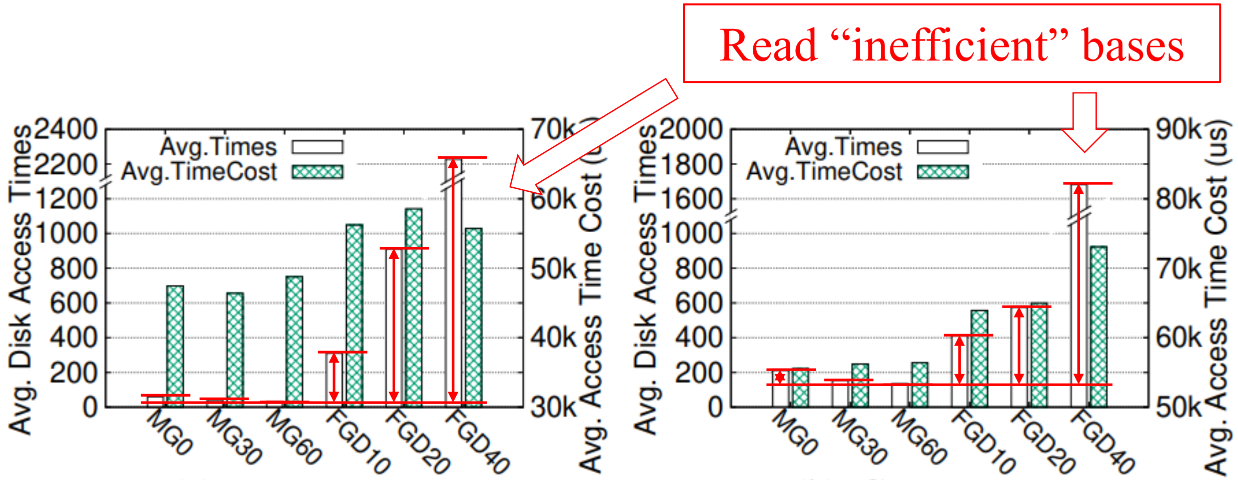
(a) WEB Dataset (b) CHM Dataset

Evaluations on the deduplication workflow

- Two parts:
    - The backup speed and statistics about accessing disks for reading bases
- Applying several parameters for FGD and MeGA
- MeGA achieves a 4.47–34.45× higher backup speed than Greedy
- Selective Delta Encoding hugely reduces disk accessing times
- Skipping more delta encoding will lead to a better speed.



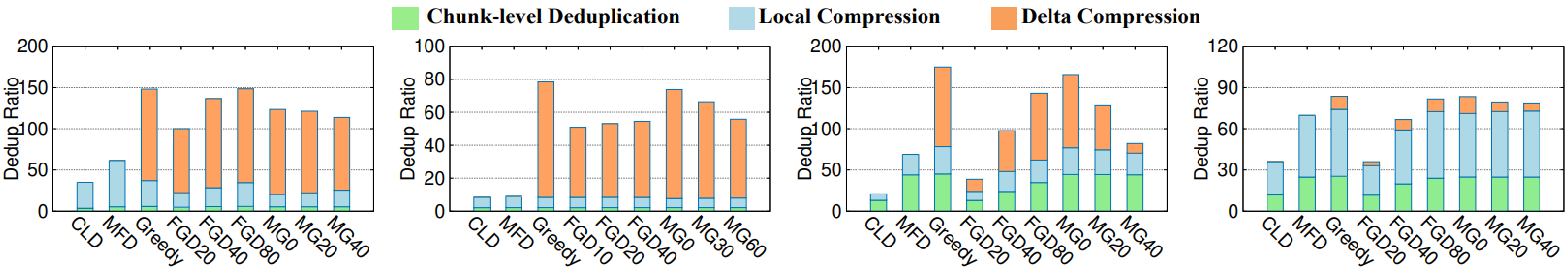
(a) WEB Dataset (b) CHM Dataset



(a) WEB Dataset (b) CHM Dataset

Evaluations on Deduplication Ratio

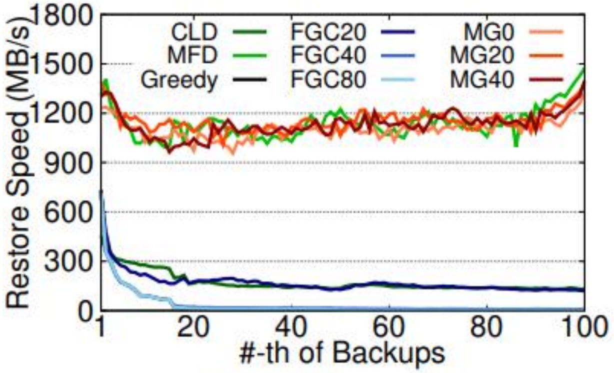
- Breakdown of deduplication ratio
- Fine-grained dedup achieves higher dedup ratio on most datasets
  - There are few similar chunks in the VMS dataset
- MeGA preserves deduplication ratio advantage
- The deduplication ratio loss caused by Selective Delta Encoding is limited



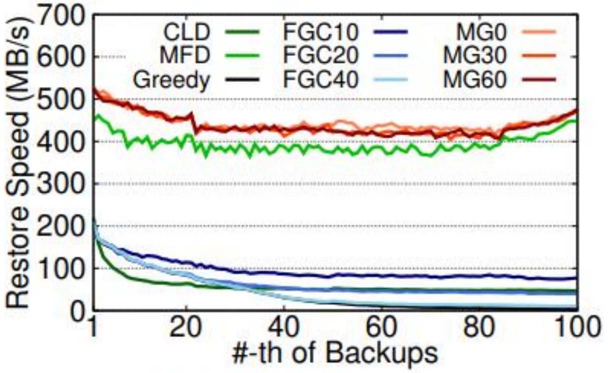
(a) WEB Dataset (b) CHM Dataset (c) SYN Dataset (d) VMS Dataset

Evaluations on the restore workflow

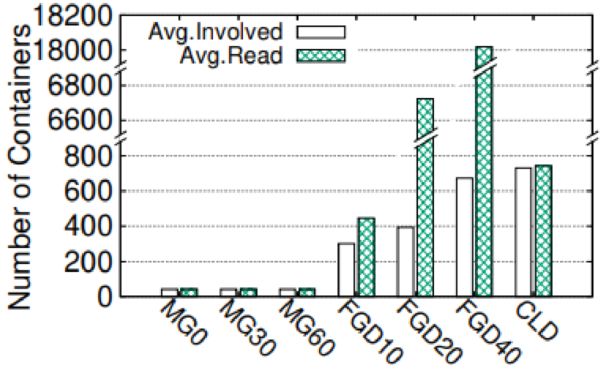
- Two parts:
  - The restore speed and statistics about accessing disks for required chunks
- MeGA achieves a 30–105× higher restore speed than Greedy.
- Our data layout hugely reduces the restore-involved containers
- Always-Forward-Reference Traversing and Delta Prewriting avoid the repeatedly accessing.



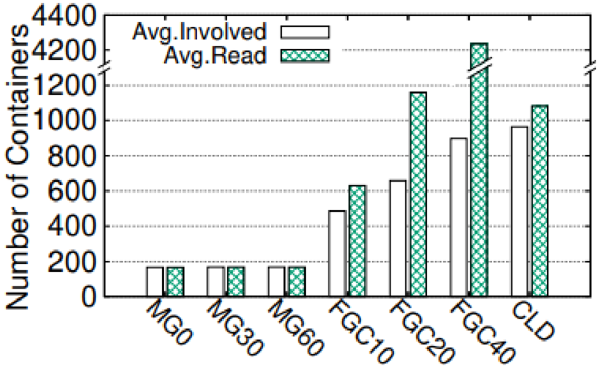
(a) WEB Dataset



(b) CHM Dataset



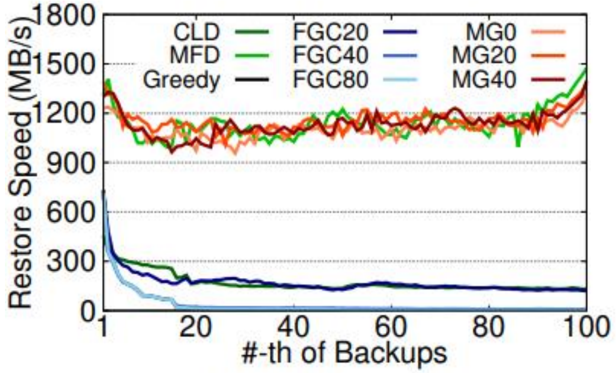
(a) WEB Dataset



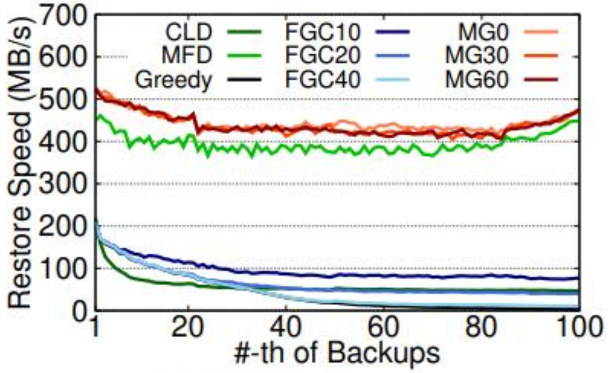
(b) CHM Dataset

Evaluations on the restore workflow

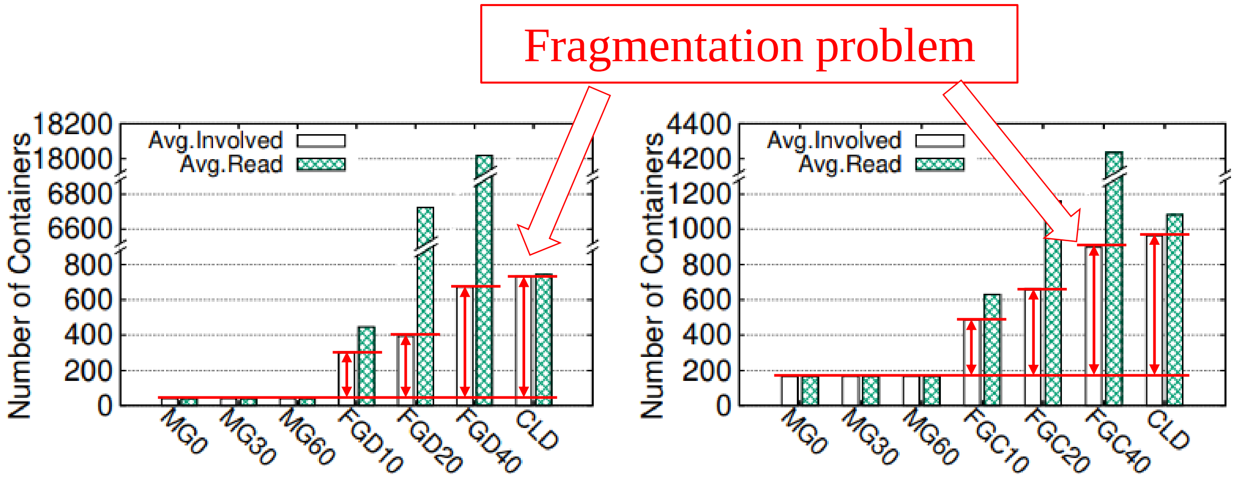
- Two parts:
  - The restore speed and statistics about accessing disks for required chunks
- MeGA achieves a 30–105× higher restore speed than Greedy.
- Our data layout hugely reduces the restore-involved containers
- Always-Forward-Reference Traversing and Delta Prewriting avoid the repeatedly accessing.



(a) WEB Dataset



(b) CHM Dataset

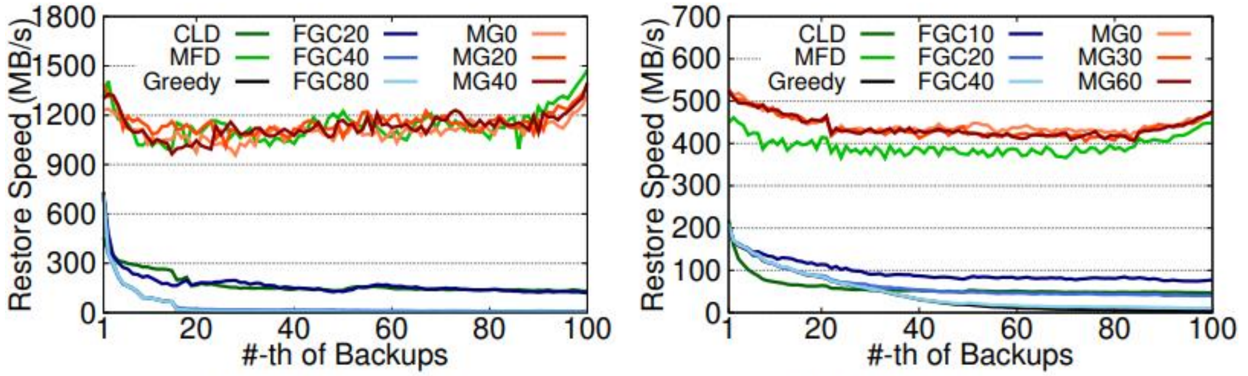


(a) WEB Dataset

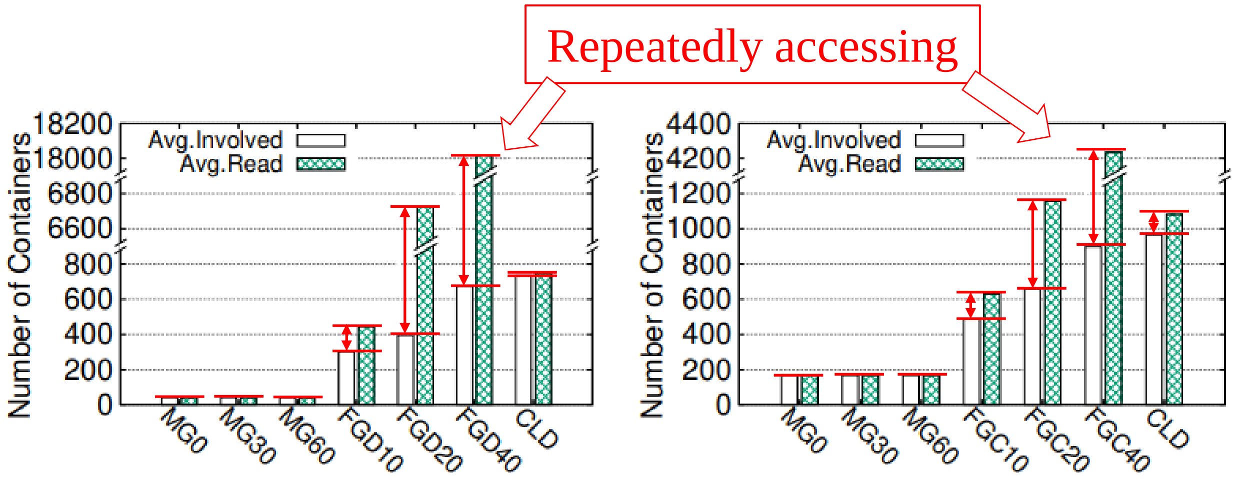
(b) CHM Dataset

Evaluations on the restore workflow

- Two parts:
  - The restore speed and statistics about accessing disks for required chunks
- MeGA achieves a 30–105× higher restore speed than Greedy.
- Our data layout hugely reduces the restore-involved containers
- Always-Forward-Reference Traversing and Delta Prewriting avoid the repeatedly accessing.



(a) WEB Dataset (b) CHM Dataset



(a) WEB Dataset (b) CHM Dataset

Conclusion

- Fine-grained deduplication introduces additional locality issues
  - Poor locality in base chunks, restore-required chunks, and delta-base pairs
- We propose three techniques to address these issues
  - Selective delta encoding
  - The delta-friendly data layout
  - Always-forward-reference traversing and delta prewriting
- Supported by these techniques, MeGA achieves:
  - 4.47–34.45× / 30–105× higher backup/restore speed than Greedy
  - Preserves fine-grained deduplication's significant dedup ratio advantage

Thank you!

Contact: xiangyu.zou@hotmail.com