The Dilemma between Deduplication and Locality: Can Both be Achieved?

Xiangyu Zou[1], Jingsong Yuan[1], Philip Shilane[2], Wen Xia[1 3], Haijun Zhang[1], and Xuan Wang[1]

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

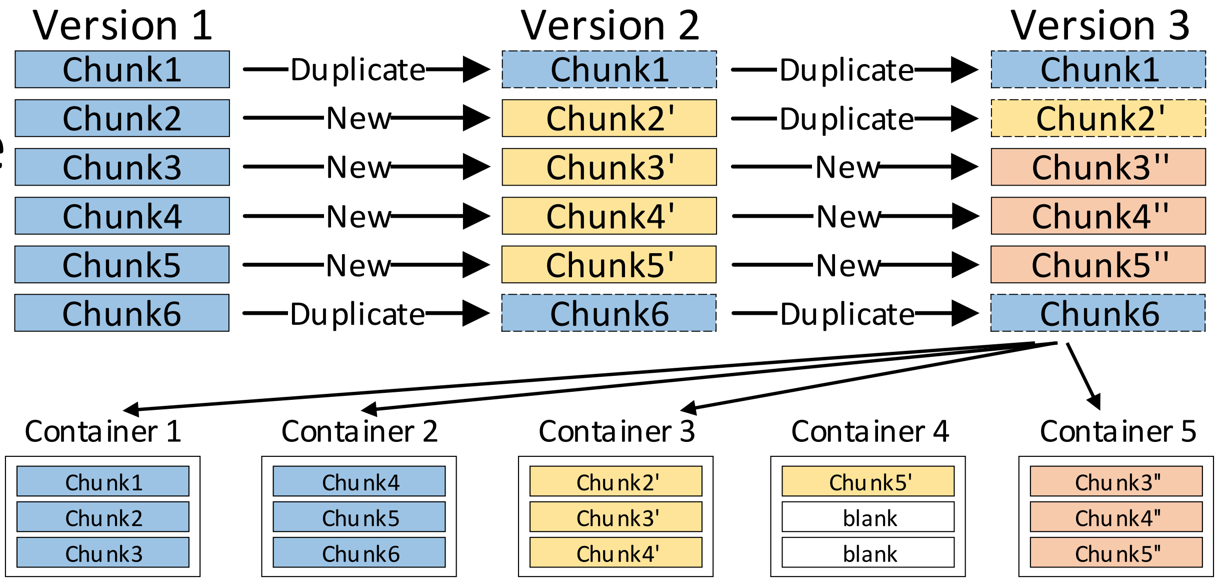
[3]Wuhan National Laboratory for Optoelectronics

FAST 2021

Background

➢ Backup workloads & Locality

- A series of backup versions (Version 1, 2, …, N)
- **10-30X** deduplication ratio
- Locality → improve performance



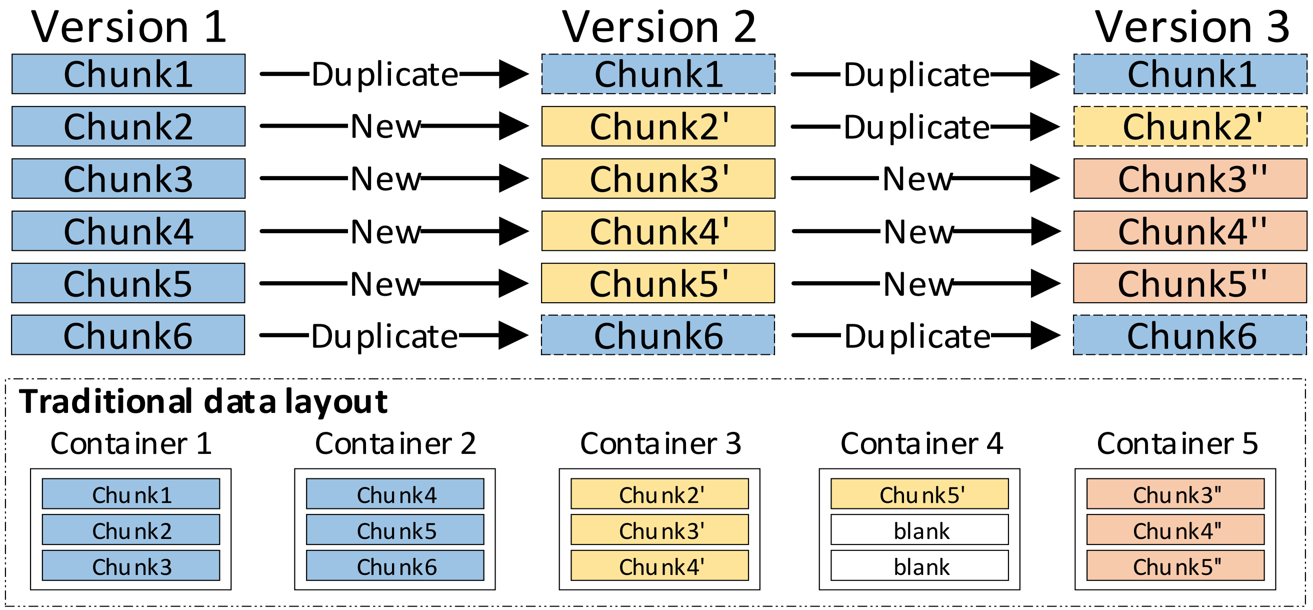
➢ Fragmentation

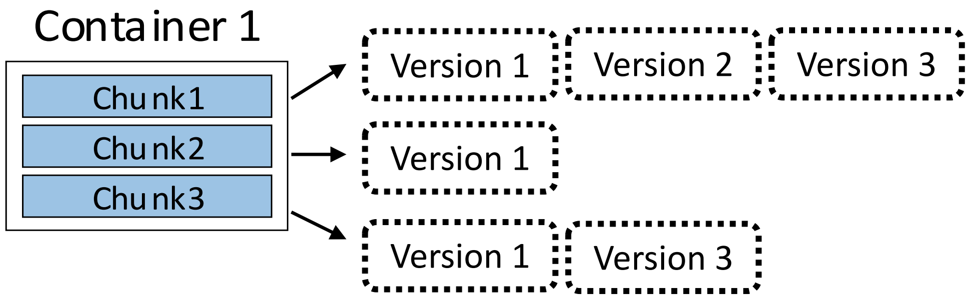
- Random access
- Read amplification
- Destroyed spatial locality after deduplication
- Garbage Collection(GC): Read amplification exists

Observation I: Read Amplification

➢ **What causes read amplification/fragmentation?**

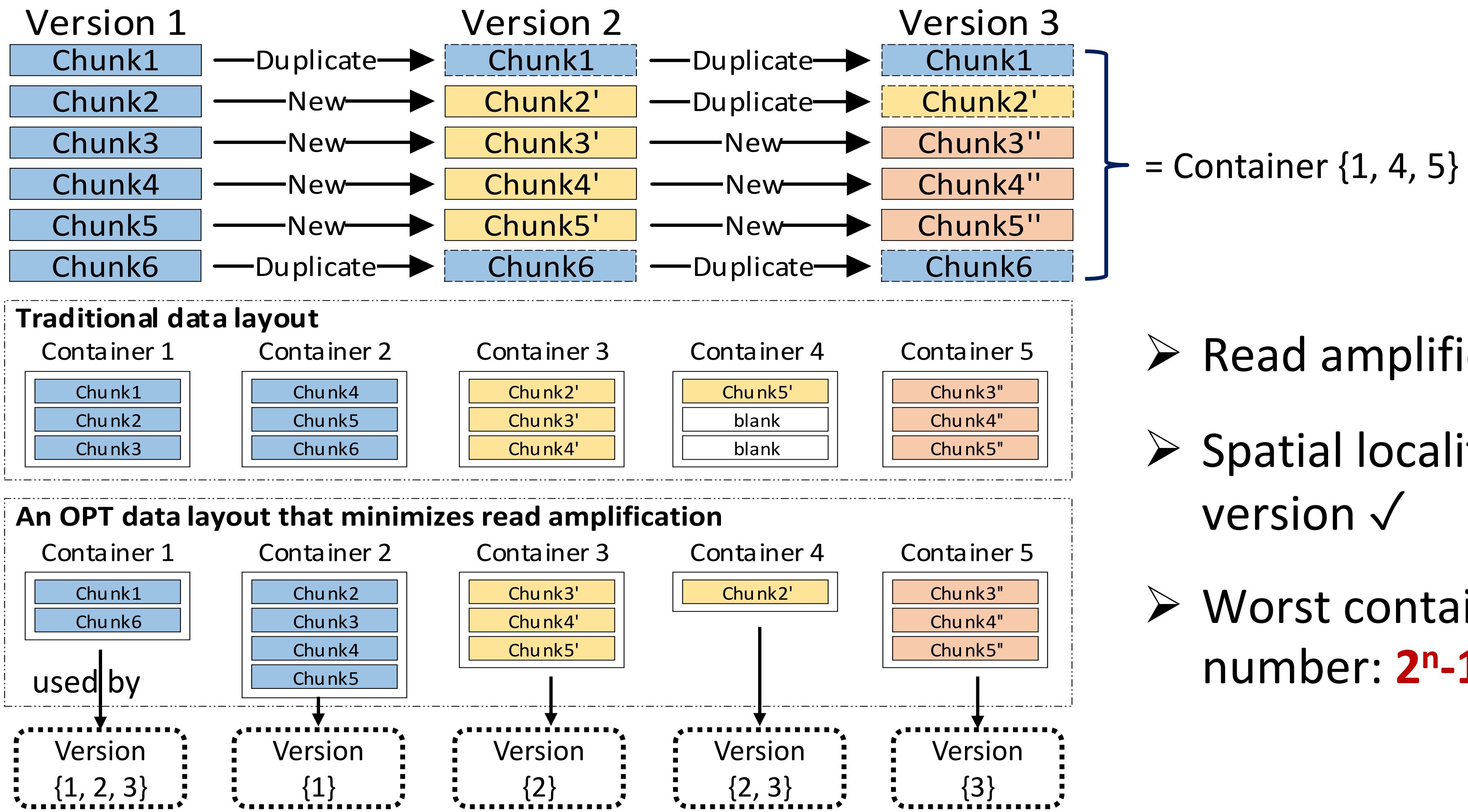
- Container-based I/O
- Chunk referenced by other versions (**Cross-version lifecycle**)





Observation II: Optimal Layout

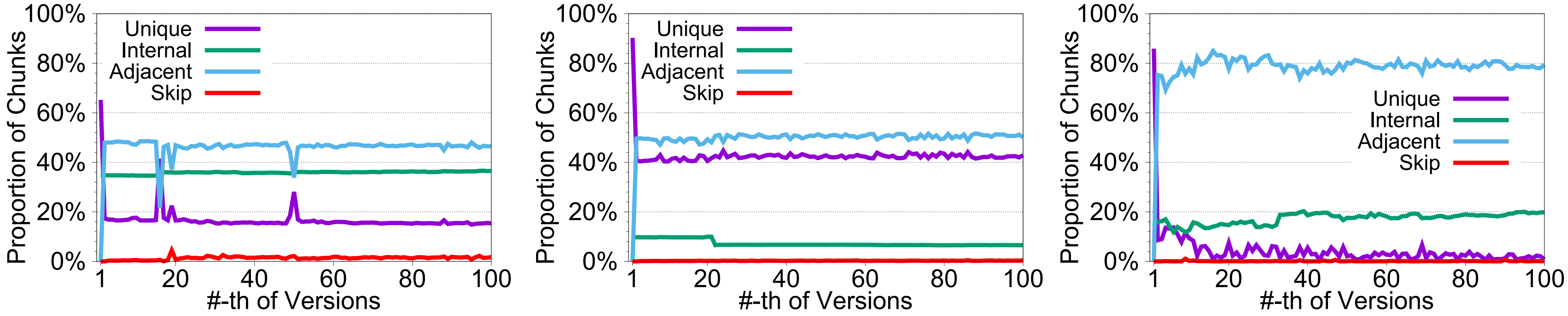
- Optimal layout: classification-based according to reference relationship



- Read amplification: 1X
- Spatial locality for each backup version ✓
- Worst containers(categories) number: **2^n-1, O(2^n)**

Observation III: Derivation of Backups

- Denote
  - ***Internal*** duplicate chunks: duplicate in B_i
  - ***Adjacent*** duplicate chunks: duplicate not in B_i but in previous version B_{i-1}
  - ***Skip*** duplicate chunks: duplicate neither in B_i not in B_{i-1} (possible in B_{i-2}, B_{i-3}, or …)
  - ***Unique*** chunks: non-duplicate



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

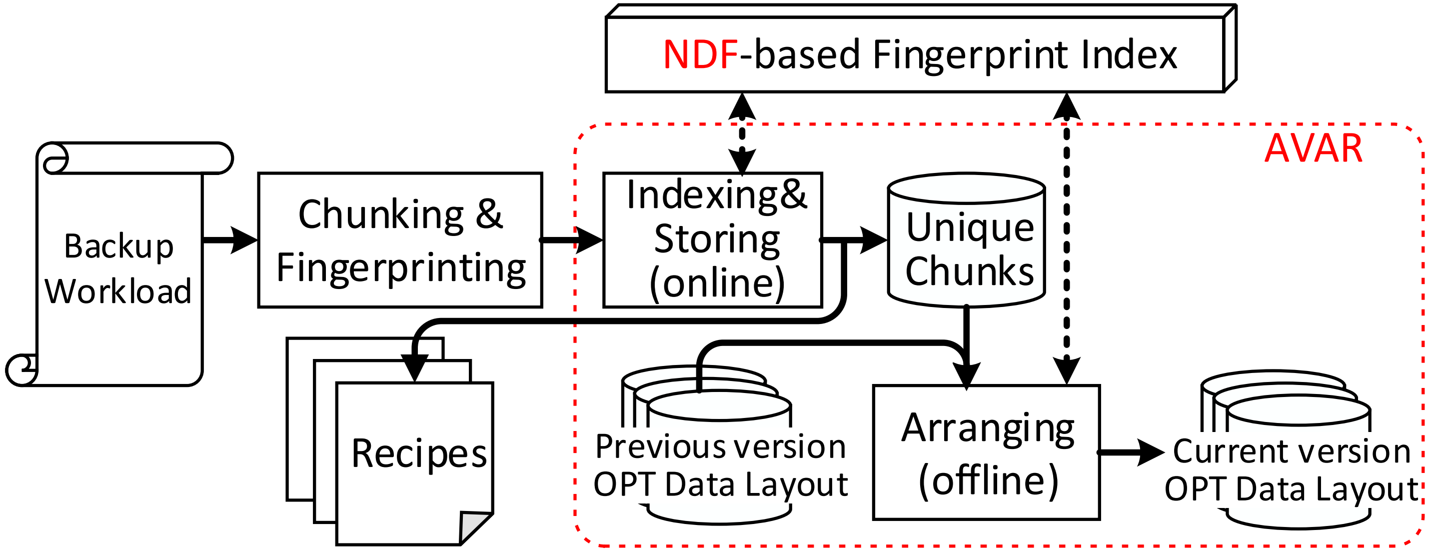
- ***Adjacent*** and ***Internal*** account for about **99.5%** in most datasets.
- ***Skip*** only consists of a small fraction (less than **0.5%**).

Motivation

- **Fragmentation/Read amplification**
  - Use **OPT** data layout
- **OPT has unacceptable numbers of containers(categories)**
  - Focus neighbor duplicates:
    - ***skip*** duplicate chunks as unique to reduce container numbers
    - Worst case: **$n(n+1)/2$, $O(n^2) \leftarrow O(2^n)$**
  - Utilize the derivation relationship of backups to reduce
- **Duplicate checking across versions**
  - Reorganize chunks between versions

Architecture

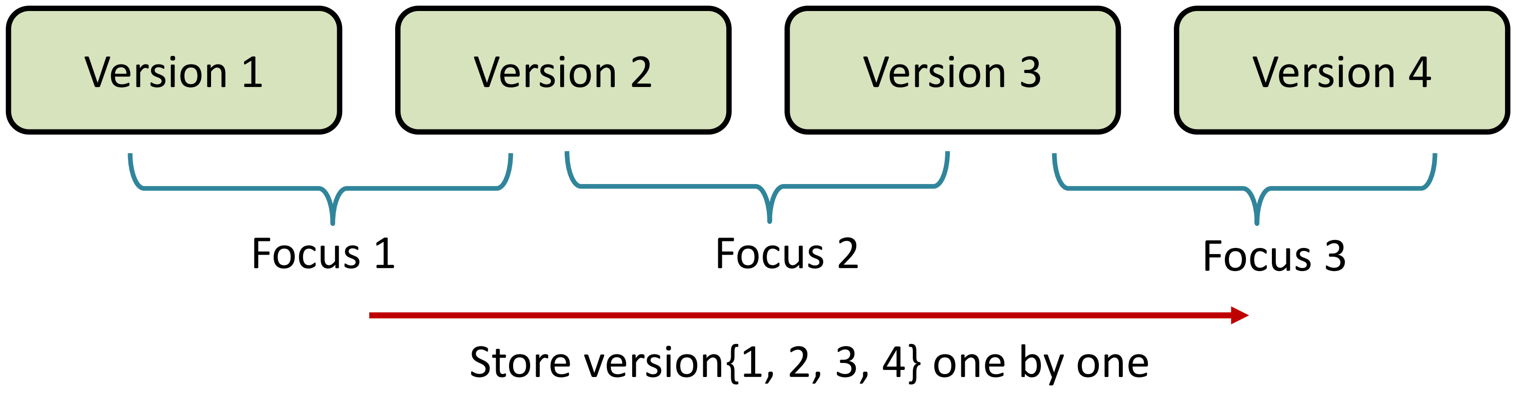
- ***MFDedup***, a management-friendly deduplication framework using **OPT** data layout, which eliminates fragmentation in order to improve restore and GC performance.
- Techniques
  - **N**eighbor-**D**uplicate-**F**ocus indexing (**NDF**)
  - **A**cross-**V**ersion-**A**ware **R**eorganization scheme (**AVAR**)



Neighbor-Duplicate-Focus(NDF) Indexing

- **NDF indexing: Neighbor-deduplicate-focus**
  - Most duplicate chunks for a backup version are from the previous version (***Adjacent***)
  - Treat ***Skip*** duplicate chunks as unique chunks instead of deduplicating them
  - For chunks in B_i: Deduplicate check against either in B_i(***internal***) or B_{i-1}(***previous***)

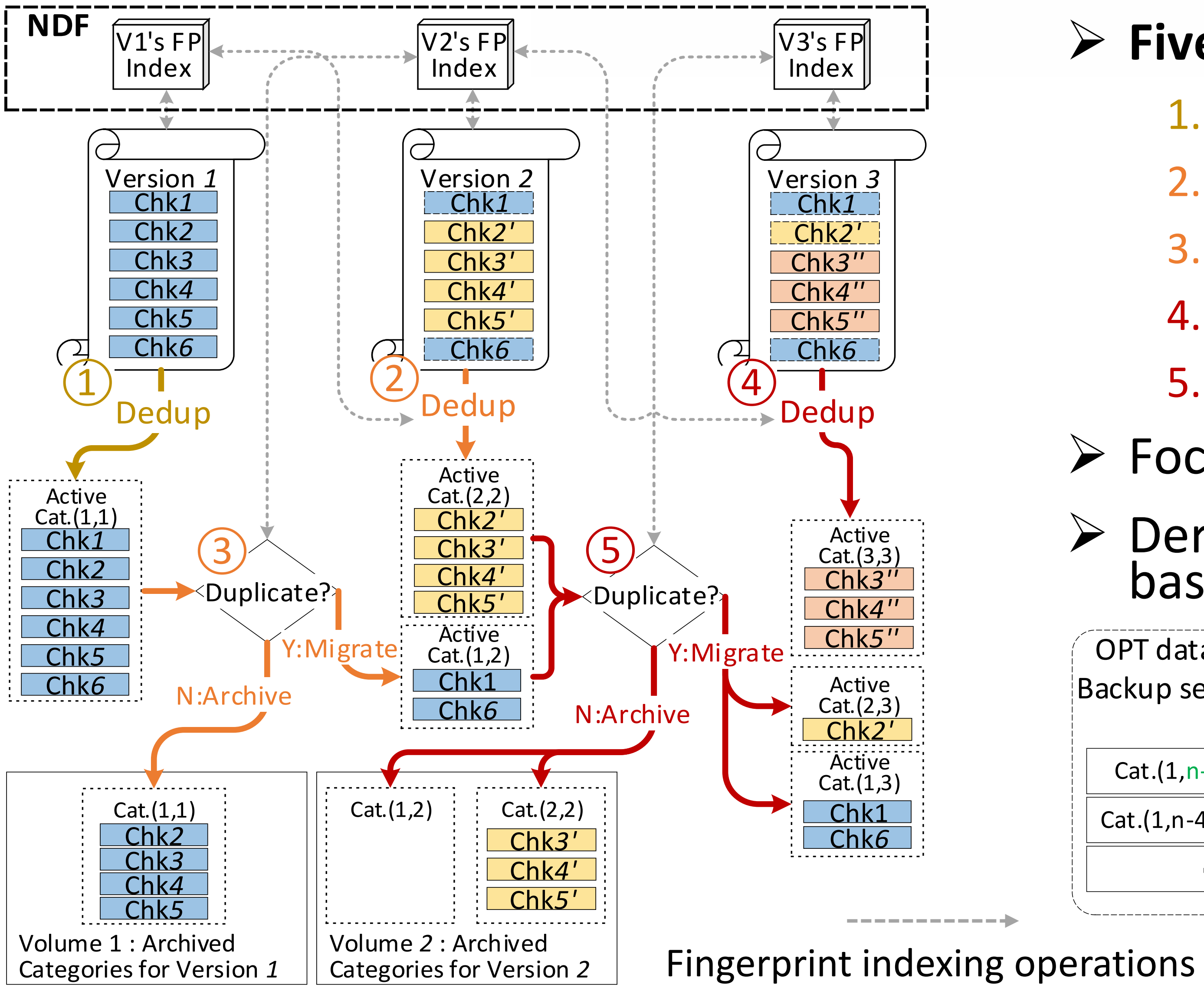
NDF Example



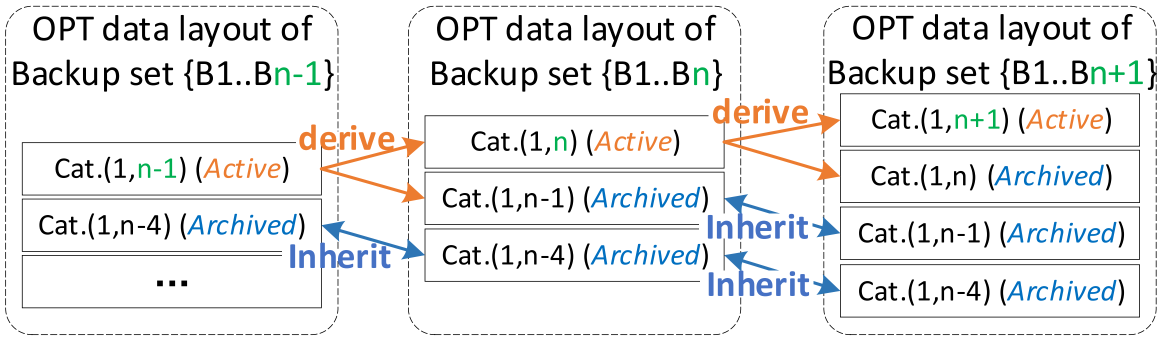
Across-Version-Aware Reorganization

- **Two stages**
  - Deduplicating stage
    - Identify unique chunks in B_i using **NDF**-based fingerprint index
    - store them into a new ***active*** container (category)
    - **Cat.(2, 4)**: chunks referenced by consecutive Versions {2, 3, 4}
  - Arranging stage
    - Iteratively update the existing **OPT** data layout with new unique chunks of the incoming version (such as B_{i+1})
    - **Archive** categories which are not referenced by current version B_i
    - Focus on ***active*** categories

AVAR Workflow

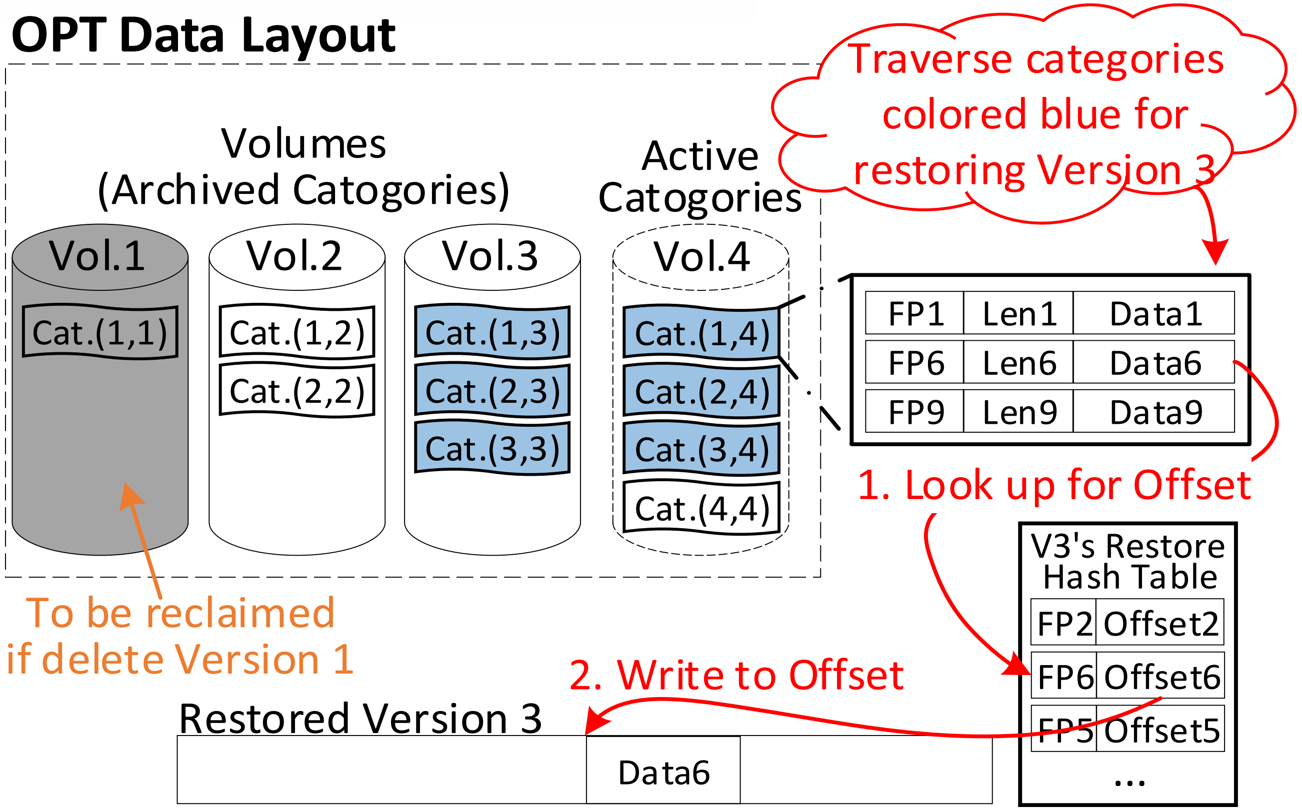


- **Five steps**
  1. Deduplicating Version 1
  2. Deduplicating Version 2
  3. Arranging Version 1
  4. Deduplicating Version 3
  5. Arranging Version 2
- Focus on active category
- Derive new active category and archive based on previous active one



Restore & Delete

- **Delete**
  - Delete version n → Delete **Cat.(n, n)**
- **Restore**
  - Traverse categories
  - Assemble offsets to form restore hash table



Experimental Setup

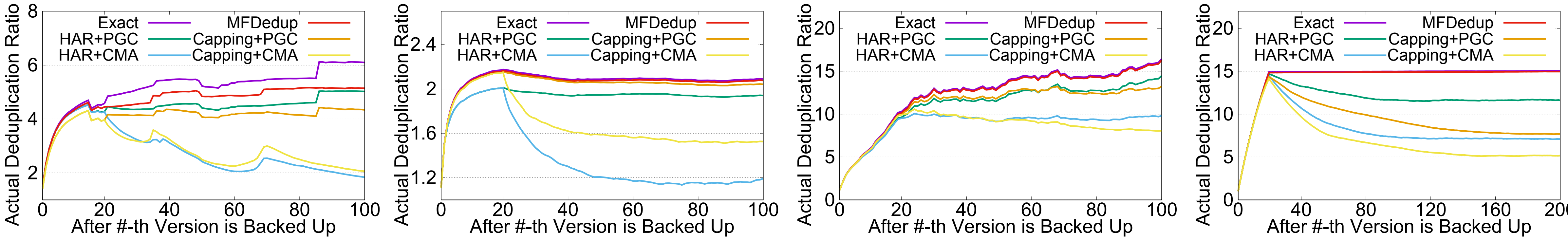
- Evaluation
  - Device: Intel D3-S4610 SSDs (User space), and 7200rpm HDDs (Backup space)
  - Policy: Retain the most recent 20 versions while GC starts from the 21^{st} version
  - Write: User space → Backup space
  - Baseline: Rewrite (**HAR** and **Capping**), GC (**Perfect GC** and **CMA**)
  - Datasets: **WEB**, **CHM**, **SYN**, and **VMS**

Name	Total Size Before Dedup	Versions	Workload Descriptions
WEB	269 GB	100	Backup snapshots of website in 2016
CHM	279 GB	100	Source codes of Chromium project
VMS	1.55 TB	100	Backups of an Ubuntu 12.04 Virtual Machine
SYN	1.38 TB	200	Synthetic backups by simulating file create/delete/modify operations

Actual Deduplication Ratio(ADF)

- **Actual Deduplication Ratio** = $\frac{\text{Total size of the dataset}}{\text{Size after running an approach}}$
  - **MFDedup** ignores ***Skip*** duplicate chunks
  - Rewriting and GC consume more storage for better restore and GC performance

- PGC: Greediest GC
- CMA: Laziest GC

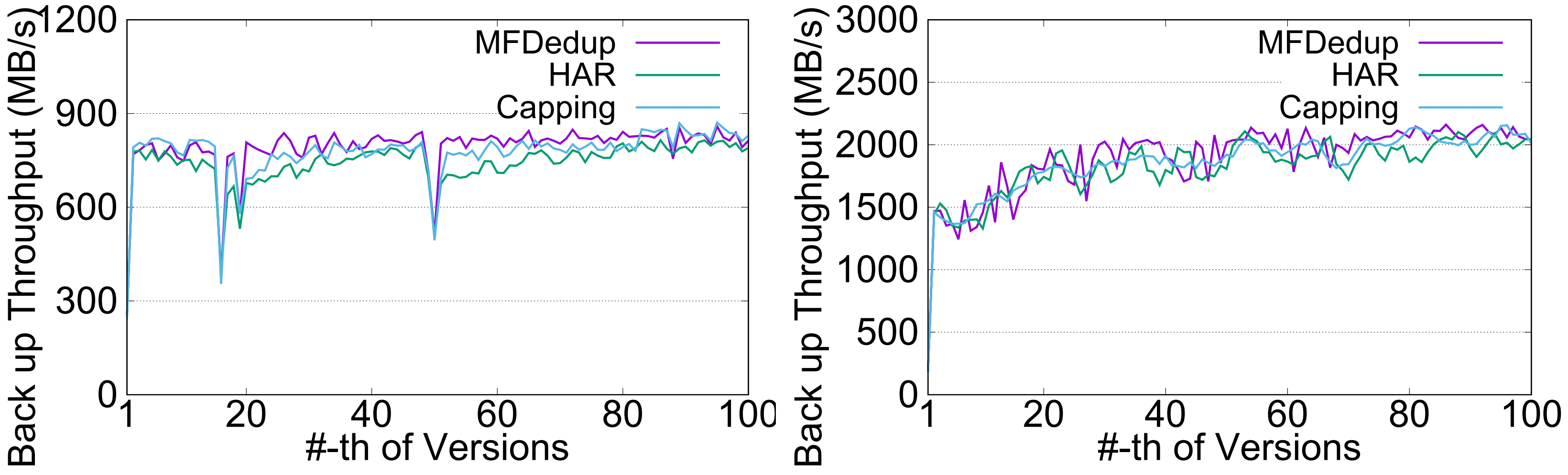


(a) WEB (b) CHM (c) VMS (d) SYN

- **MFDedup** achieves **1.12-2.19X** higher **ADR** compared with other approaches due to the **OPT** data layout and small impact on ignoring ***Skip*** duplicate chunk.
- Higher GC leads to fewer unreferenced chunks.
- Rewrite reduces deduplication ratio.

Backup Throughput

- Offline process is not considered, including **arranging** and **GC**
- To minimize the performance impact of reading datasets, backup up datasets from **ramdisk**
- Evaluate two selected datasets only due to space limit

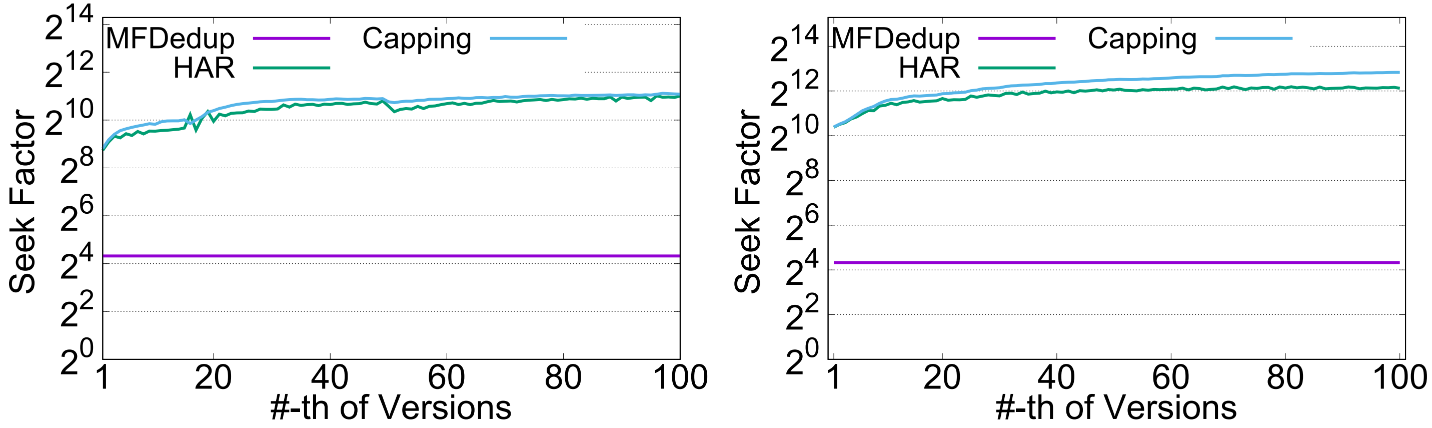


(a) WEB (b) VMS

- **MFDedup** does not sacrifice backup throughput to achieve the other benefits.

Seeking Overhead

- ➢ Seek Number: the number of seek operations required for reading containers/volumes on disk devices

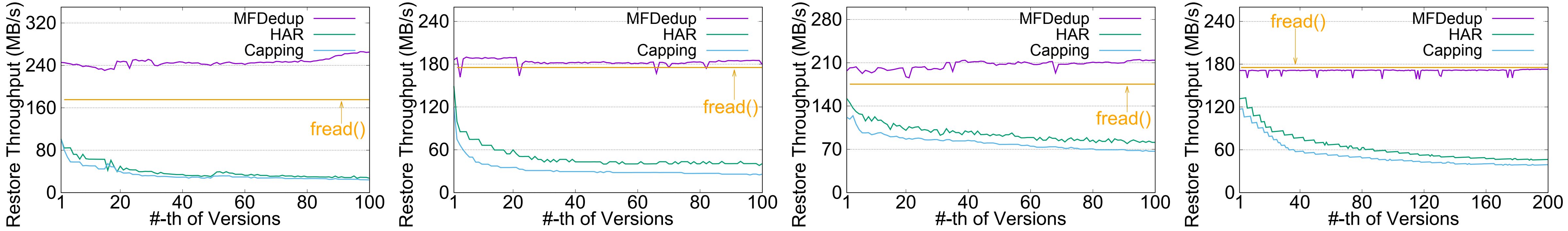


(a) WEB (b) VMS

- ➢ **MFDedup** reduce seek number greatly since it groups several archived categories into one big and sequentially written volume.
- ➢ **Capping** and **HAR** need more seek operations due to the existence of fragmentation.

Restore Throughput

- *fread()*: the sequential throughput of the backup device

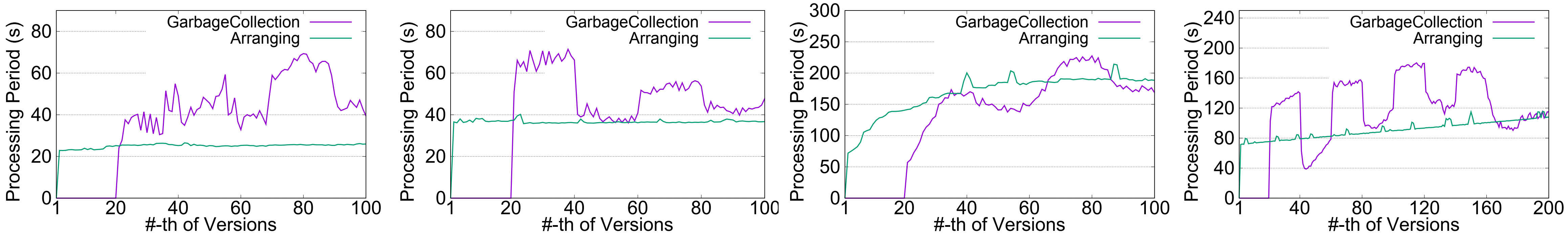


(a) WEB (b) CHM (c) VMS (d) SYN

- **MFDedup** achieves up to 11.64× (WEB), 4.54× (CHM), 2.63× (VMS) and 3.73× (SYN) higher than **HAR**.
- Fragmentation still exists in **HAR** and **Capping** and becomes worse with higher version.

Time Cost: MFDedup vs GC

- **Arranging**: running between 20 versions
- **GC**: start GC from 21st version (retaining 20 versions) – excluding selection phase



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

- **Arranging**'s total processing period is only **45% (WEB)**, **37% (CHM)** and **25% (SYN)** of **GC**'s total processing time on average.
- Changing the same region make **GC** very easy in **VMS** which makes **Arranging** takes 9% longer than GC.

Conclusion

- **MFDedup**, a management-friendly deduplication framework using **OPT** data layout, which eliminates fragmentation in order to improve restore and GC performance.
  - Focus on neighbor-duplicate indexing
  - Ignore ***Skip*** duplicate chunks to reduce greatly categories based on **OPT** layout
  - Arrange to update **OPT** layout
- Actual Deduplication Ratios (**1.12-2.19X** higher)
- Restore throughput (**2.63-11.64X** higher)