

Coupling Right-Provisioned Cold Storage Data Centers with Deduplication

Liangfeng Cheng¹, Yuchong Hu¹, Zhaokang Ke¹, and Zhongjie Wu²

¹Huazhong University of Science and Technology

²Alibaba Group

ICPP 2021



Cold Storage

➤ Property 1: **Vast quantity**

- IDC predicts that the global data size will grow from 33 ZB in 2018 to 175 ZB by 2025. [The digitization of the world from edge to core, 2018]
- Facebook points out that more than 80% of data are cold. [f4, OSDI'14]

➤ Property 2: **Rarely read but not unacceptable long access**

- Increasingly investing in big data analytics to derive insights in seconds. [Borovica-Gajić, VLDB'16]

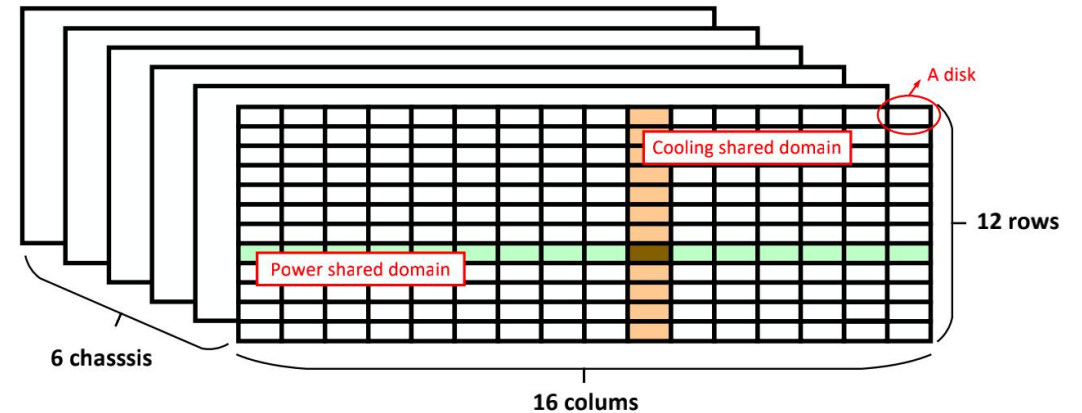
➤ How to reduce the cost of storing cold data with acceptable long access?

- **Right-provisioning** for reducing the cost of power and capital in data centers
- **Deduplication** for reducing the storage overhead

Right-provisioning

➤ New rack design in Pelican [OSDI'14]

- Grouping all disks, only **one group** is active at any given time
- **Group switch** for changing active group currently



➤ Acceptable second-long access

- HDD-based systems
- Group switch takes 8 seconds
- **single file is stored into single group, considered as Single File Single Group (SFSG) constraint**

*1,152 disks in 6 chassis, each of which has 192 disks with 12 (rows)*16 (columns)*

Deduplication

➤ Cold data has much redundancy

- Backup is a compelling usage model in cloud cold storage [Muthitacharoen, SOSP'01]
- Backup data includes many redundant versions of files

➤ The chunk-level deduplication splits the input data stream into multiple chunks, identifies each chunk by fingerprints, eliminates duplicate chunks, and stores only unique chunks

- **Chunking**
- **Fingerprinting**
- **Indexing**
- **Storage management**

Backup Datasets

- Four widely used real-world backup datasets
- **Linux**, <http://www.kernel.org/>
 - **GCC**, <http://ftp.gnu.org/gnu/gcc/>.
 - **VMDK**, which is created by a virtual machine with Ubuntu (v.16.04) running common tasks
 - **RDB**, consists of many Redis snapshots (5GB with **1% change rate** on average from YCSB [SoCC'10])

Datasets	Linux	GCC	VMDK	RDB
Total Size (GB)	384.76	32.29	1569.63	1154.45
Number of Files	35.5×10^6	5.4×10^6	86	216
Deduplication Ratio	55.9	8.29	73.4	13.1

Main Challenge and Method

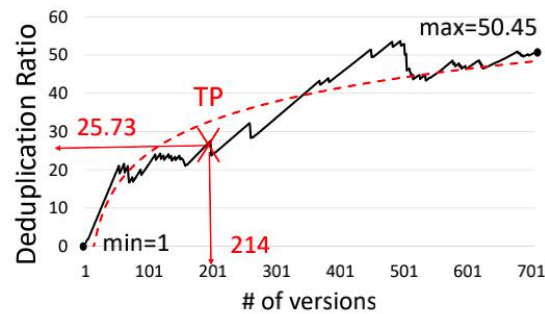
- **Classical deduplications have no idea of “group” of right-provision**
 - **A file would be deduplicated across multiple groups**
 - Violate SFSG constrain and trigger group switches
 - Method: **simple intra-dedup**
- **Lower deduplication ratio**
 - The simple intra-dedup has much lower deduplication ratio than that of classical-dedup
 - Method: **manipulate versions across groups**

	# of versions	Data size after dedup.
Classical-dedup	704 together	6.89 GB
Simple intra-dedup	58 or 59 per group	53.27 GB

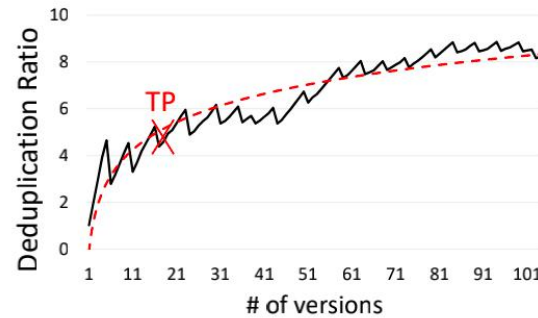
Observations

➤ Turing-point (TP)

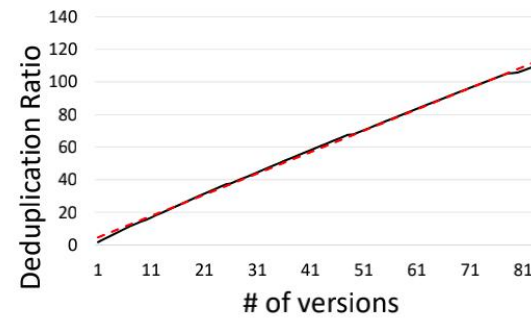
- Differentiate the two increasing parts (i.e., before TP and after TP).
- $50\%(\text{min} + \text{max})$ as default, where min \rightarrow the deduplication ratio of the first version and the max \rightarrow that of the entire version



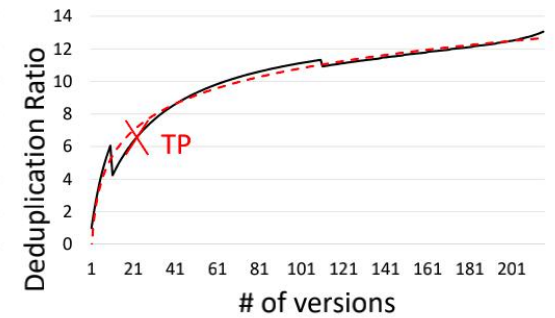
(a) Linux



(b) GCC



(c) VMDK



(d) RDB

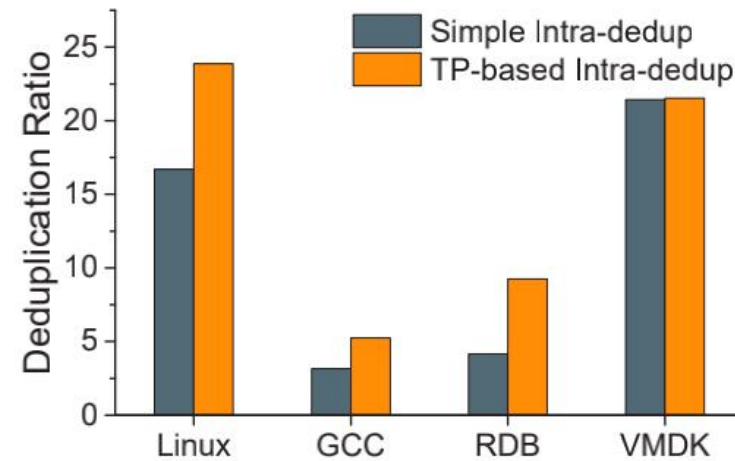
Observation 1

➤ Non-linear increasing deduplication ratio

- Single intra-dedup is distributed version in average
- For most dataaets, TP-based intra-dedup can improve the deduplication ratio significantly compared to the simple intra-dedup

# of version of Linux	Simple	TP-based
Total versions	704	704
G_1	176	214
G_2	176	214
G_3	176	214
G_4	176	62

(a) Distribution of the Linux dataset



(b) Deduplication ratios

Observations 2

➤ Long tail after the turning point

- TP-based intra-dedup cannot be comparable with that of classical-dedup after a large number of versions.
- Intra-dedup leaves a number of redundant data between groups
- The inter-group redundancy increases with the number of groups

	Data size before or after dedup. (GB)			
	Linux	GCC	VMDK	RDB
Total size before dedup.	384.76	32.29	1569.63	1154.45
Classical-dedup	6.89	3.93	21.37	88.42
Intra-dedup across 2 groups	12.91	6.52	40.78	160.37
Intra-dedup across 4 groups	23.04	10.23	73.27	277.73
Intra-dedup across 8 groups	39.99	13.63	127.83	423.55
Intra-dedup across 12 groups	53.27	16.72	174.72	535.14

Analysis and set theory based Model

➤ Many identical and similar files between groups

- **Identical files**, whose all chunks simultaneously exist in multiple groups
- **Similar files**, whose most chunks (i.e., λ out of total chunks) simultaneously exist in multiple groups
- **For Linux and GCC, identical files dominate all the files**
- **For VMDK and RDB, no identical file but similar files account for more than 90%**

		Dedup. ratios			
		Linux	GCC	VMDK	RDB
Identical files		93.00%	87.00%	0.00%	0.00%
Similar files	$\lambda = 0.60$	0.03%	0.29%	93.06%	100.00%
	$\lambda = 0.65$	0.03%	0.31%	93.06%	100.00%
	$\lambda = 0.70$	0.03%	0.31%	93.06%	100.00%
	$\lambda = 0.75$	0.03%	0.31%	93.06%	100.00%
	$\lambda = 0.80$	0.03%	0.31%	93.06%	100.00%
	$\lambda = 0.85$	0.02%	0.24%	93.06%	99.54%
	$\lambda = 0.90$	0.02%	0.18%	93.06%	99.07%
	$\lambda = 0.95$	0.01%	0.10%	91.67%	98.61%

➤ Can we find and eliminate these identical and similar files efficiently?

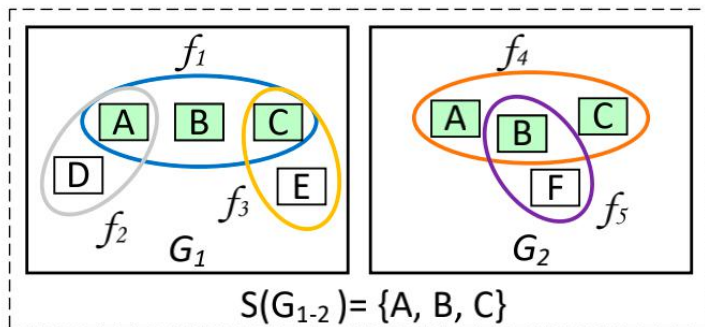
Analysis and set theory based Model

➤ Notations

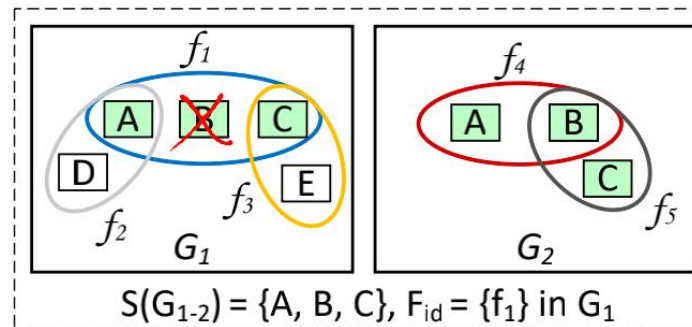
- F_i and G_j indicate a file and a group respectively
- $S(G_{1-2})$ indicates the set of all same chunks between G_1 and G_2
- F_{id} and F_{si} indicates all identical and similar files of a group respectively

➤ Three Theorems

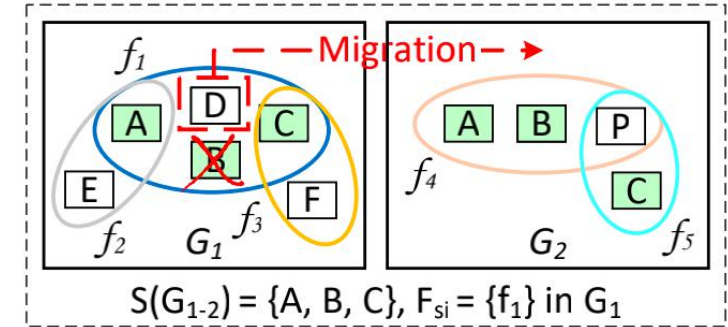
- **Identify identical and similar files**
- **Give redundant chunks for identical files**
- **Give redundant chunks and migrating chunks for similar files**



(a) Identifying identical/similar files
(Theorem 1)



(b) Chunk elimination for identical files
(Theorem 2)



(c) Chunk elimination and migration for similar file
(Theorem 3)

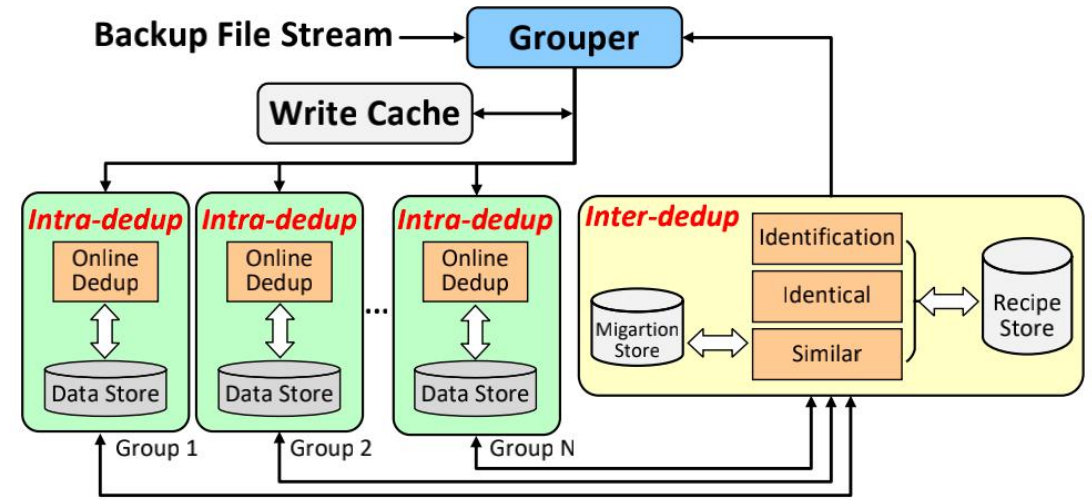
Our Contributions

- We are the first to combine right-provisioning with deduplication
 - Give observations based on TP to characterize the deduplication ratio with the number of versions
 - Give a theory based model to characterize inter-group redundancies
- We design DeCold
 - Propose an online TP-based intra-dedup
 - Propose an additional offline set theory model based inter-dedup
 - Design three improvements for inter-dedup efficiency
- We implement DeCold atop Destor [FAST'15]
 - Evaluations conducted on four real-world backup datasets demonstrate the deduplication efficiency and access performance
 - Release DeCold at <https://github.com/yuchonghu/decold>

DeCold Architecture

➤ Five modules

- N groups' **disks**
- a **Grouper**
- a **Write cache**
- N **Intra-dedup** including Data Store
- an **Inter-dedup** including **Identification**, **Identical**, **Similar** submodules, Recipe Store, and Migration Store



➤ Satisfy the SFSG constraint

- Each group has an independent recipe for online TP-based intra-dedup
- Grouper module assigns each file a *fid* and the currently spinning group *gid*, and then ensures a one-to-one mapping, i.e., $fid \rightarrow gid$.

Intra-dedup Module

➤ Separating metadata from data

- Storing chunks in Data Store inside groups while keeping recipes in Recipe Store outside of groups

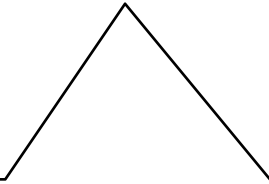
➤ Enabling TP-based intra-dedup

- If not know TP in advance, we can use the simple method, and **adjust the number of versions via calculating the deduplication ratio in real time**
- If current active group does not have enough versions, Write Cache module will hold these versions temporarily until the number of versions achieves TP

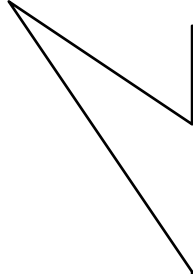
Inter-dedup Module

➤ Identification submodule: Identification of identical and similar files

- Finding inter-group chunks
- Realizing Theorem 1



1. Finding the f_{id} and f_{si} via Theorem 1

- 
1. Obtaining all chunks' fps of G_1 and G_2 via reading their Recipe Stores.
 2. Calculating $S(G_{1-2})$ via sorting all fps of G_1 and G_2 , and traversing then in a two-pointer way

Inter-dedup Module

➤ Identical submodule: Chunk elimination for identical files

- **Realizing Theorem 2**
- **Eliminating chunks offline**
- **Updating Grouper and Recipe Store**

1. Via Theorem 2, getting the eliminated chunk
 $S(G_1) - S(F_1 - F_{id})$

1. Updating the mappings of the involved identical files in Grouper
2. Updating the recipes of these files in Recipe Store

1. Removing these chunks until G1 (or G2) turns to be active

Inter-dedup Module

➤ **Similar submodule: Chunk elimination and migration for similar files**

- **Realizing chunk elimination in Theorem 3**
- **Realizing chunk migration in Theorem 3**
- **Eliminating and migrating chunks offline**
- **Updating Grouper and Recipe Store**

1. Via Theorem 3, getting the eliminated chunk $S(G_1) - S(F_1 - F_{si})$

1. Via Theorem 3, getting the migrated chunk $S(F_{si}) - S(F_{si}) \cap S(G_{1-2})$

1. Similar to identical files;
2. For migrated files, computing their fps, inserting them into the Recipe Store as well as storing their values into containers in Data of new group

1. Reading from G1's Data Store to Migration Store when G1 spins up
2. Writing from Migration Store to G₂'s Data Store when G₂ spins up

Improving Inter-dedup Efficiency

- Random-pairs for extending from two to N groups
 - Randomly divide N groups into $N/2$ pairs of 2-group inter-dedup
- Filtering for reducing scale of $S(G_{1-2})$
 - borrow the idea of the fact [Lillibridge, FAST'19] that two files sharing the same representative fps are likely to be the identical files
- Starting-size for ignoring small-size files
 - define a starting-size to perform inter-dedup to ignore these files whose file sizes are smaller than the starting-size

Implementation and Setup

➤ Implementation

- **3K SLoC in C on Linux atop Destor [FAST'15]**
- Grouper module, which is implemented by **RocksDB** to store mappings
- 12 intra-dedup modules

➤ Setup

- Intel(R) Xeon(R) Gold 5117 CPU @ 2.00 GHz, 256GB RAM and 26 1TB 3.5" SATA disks, and runs Ubuntu-16.04
- **varied-sized chunks** via Content-Defined Chunking (CDC) , 4KB-size average chunks, 4MB-size containers, and no rewriting algorithm
- Starting-size = 16KB, and $\lambda = 0.8$ as default

Metadata Analysis

➤ Metadata kinds

- Recipe Store, Grouper, Migration Store

➤ Results

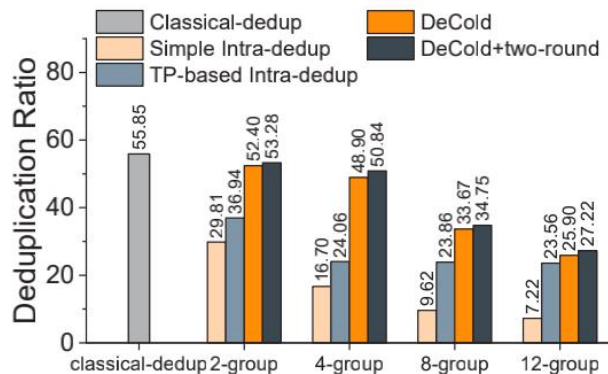
- The metadata size can be negligible compared to the data size, i.e., for VMDK, only taking up to 0.71% of the data size
- The size of Recipe store is linearly related to the data size of datasets
- The size of Grouper is linearly related to the number of files

		Linux	GCC	VMDK	RDB
Recipe Store	Metadata	2.2 GB	390.8 MB	3.5 KB	6.7 KB
	Recipes	3.8 GB	382.3 MB	9.8 GB	8.0 GB
Grouper		2.1 GB	406.5 MB	70.6 MB	70.6 MB
Migration Store		52.1 MB	126.5 MB	1.3 GB	18.1 GB

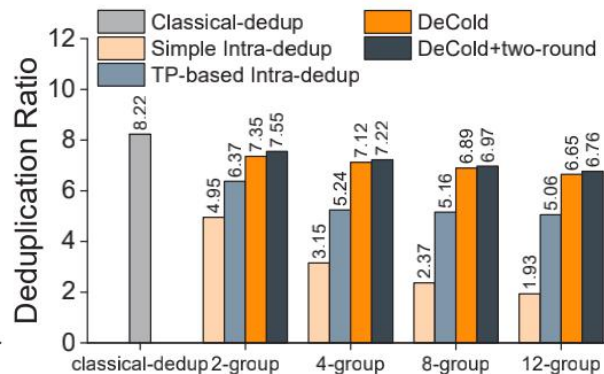
Experiment for Deduplication Ratio

➤ DeCold achieves comparable deduplication ratios

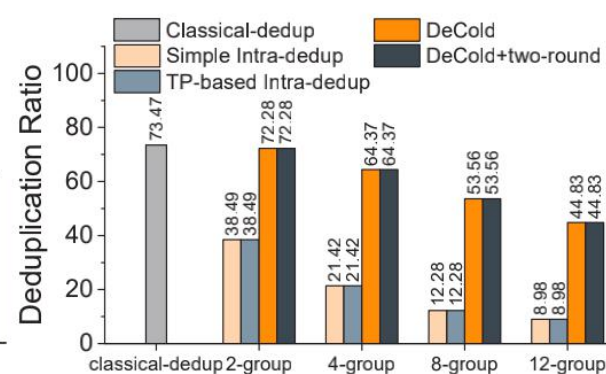
- TP-based intra-dedup realizes up to 2.5× (when RDB across 12-group) compared to the simple intra-dedup
- DeCold improves the deduplication ratio of simple intra-dedup by up to 87.8% for VMDK
- DeCold's deduplication ratio across 2-group can achieve 98.4% of classical-dedup for VMDK



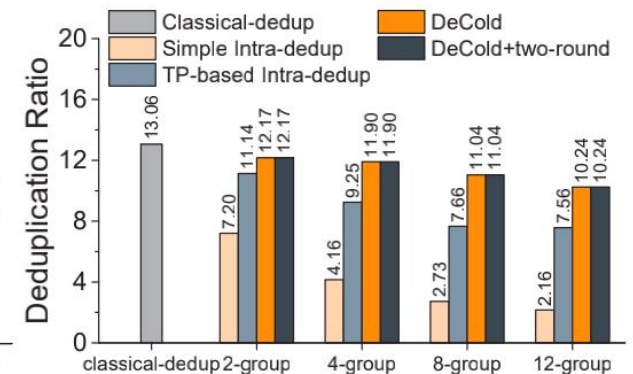
(a) Linux



(b) GCC



(c) VMDK

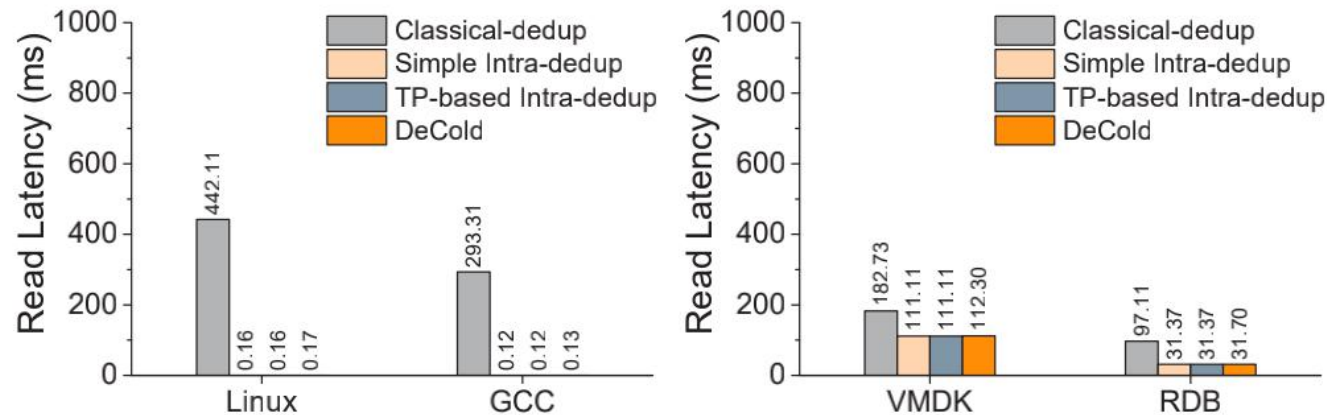


(d) RDB

Experiment for Single File Read latency

➤ DeCold achieves acceptable single file latency

- Simple intra-dedup, TP-based intra-dedup and DeCold can perform a file read by reading its chunks satisfying the SFSG constraint
- We simulate classical-dedup and find that it incurs higher file latency

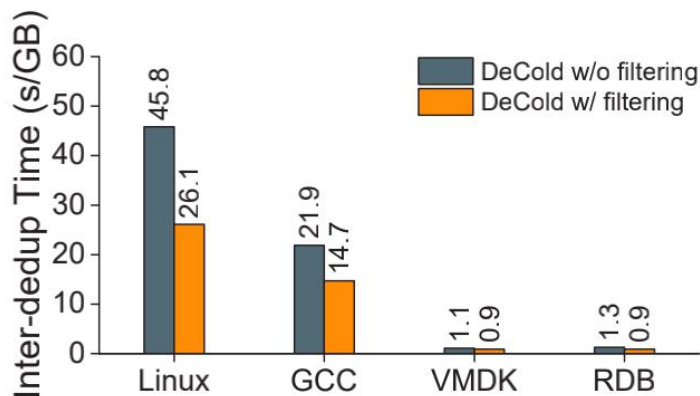


(a) Linux and GCC across 2-group (b) VMDK and RDB across 12-group

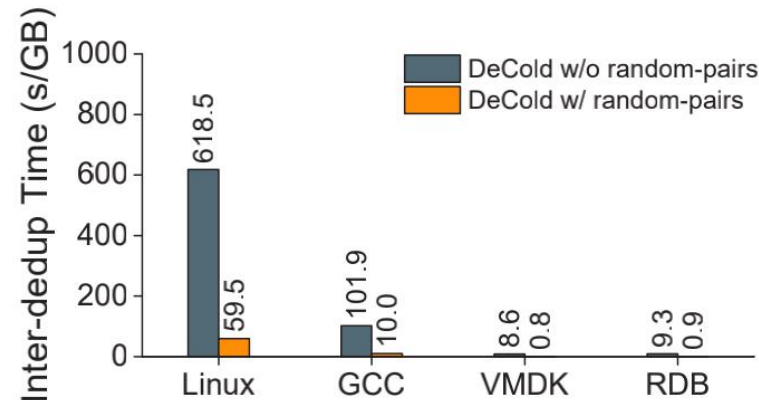
Experiment for Inter-dedup

➤ Three improvements enhance inter-dedup performance

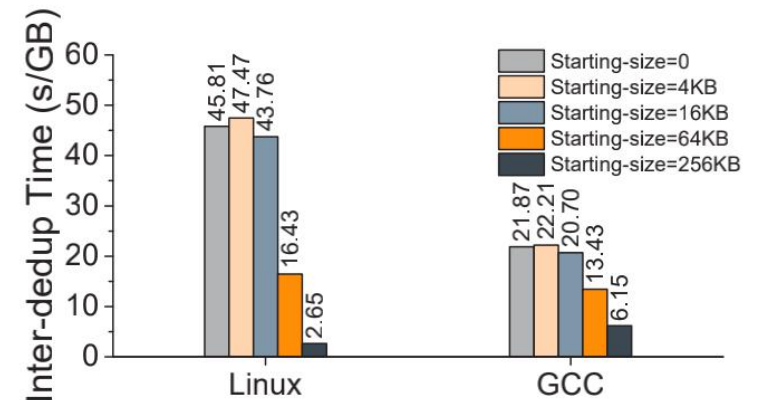
- For Linux, DeCold with filtering reduces the inter-dedup time by 43.0% compared to that of without filtering across 2-group
- for GCC, DeCold with random-pairs reduces the inter-dedup time by 90.2% compared to that of without random-pairs across 12-group
- DeCold with starting-size reduces the inter-dedup time by up to 94.2% (when the starting-size = 256KB in Linux)



(a) Filtering across 2-group



(b) Random-pairs across 12-group



(c) Starting-size across 2-group

Conclusion

- We are the first to combine right-provisioning with deduplication, propose an online TP-based intra-dedup and an additional offline model-based inter-dedup
- We design and prototype DeCold based on the above intra-dedup and inter-dedup as well as inter-dedup improvements
- Testbed experiments on four real-world backup datasets demonstrate DeCold's deduplication efficiency and acceptable read performance

DeCold prototype: <https://github.com/yuchonghu/decold>

Contact: lenfungcheng@hust.edu.cn, yuchonghu@hust.edu.cn

Thank You!
Q & A