23th USENIX Conference on File and Storage Technologies
February 25–27, 2025

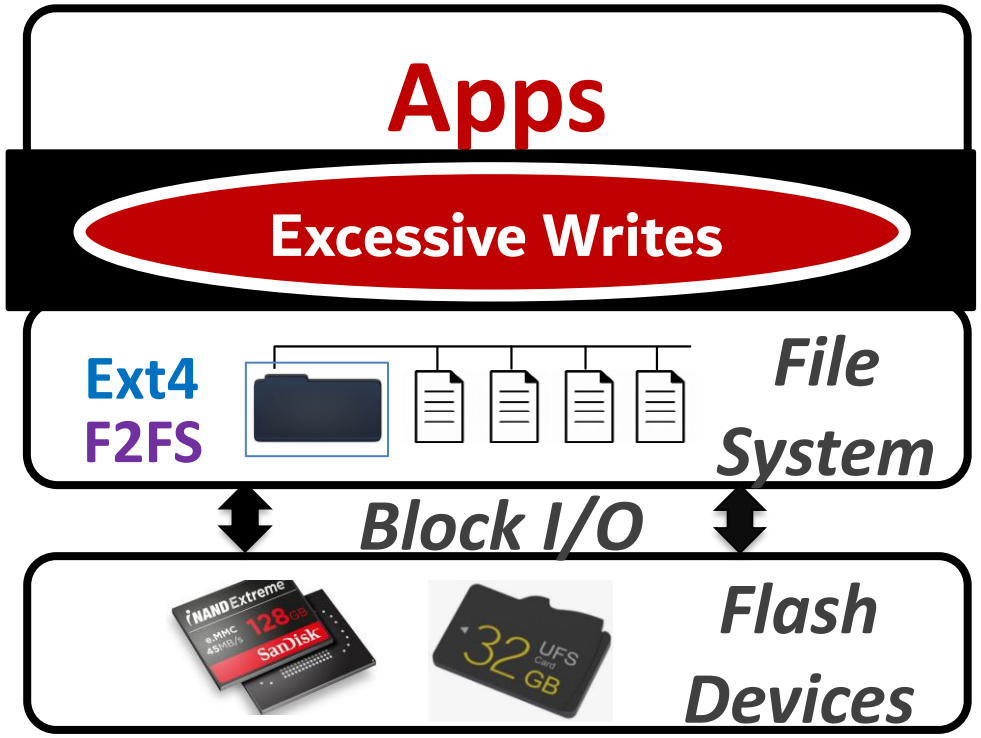
MedFS: Pursuing Low Update Overhead via Metadata-Enabled Delta Compression for Log-structured File System on Mobile Device

Chao Wu [§], *Cheng Ji* [§], *Li-Pin Chang* [※], *Zongwei Zhu* [¶], *Congming Gao* [*], *Weichao Guo* [•], *Chao Yu* [•], *Yanzhi Wang* [#]

[§] Nanjing University of Science and Technology [※] **NYCU, Taiwan**
[*]Xiamen University [¶]University of Science and Technology of China [#]Northeastern University [•]Guangdong Oppo Mobile Telecommunications Corp., Ltd.

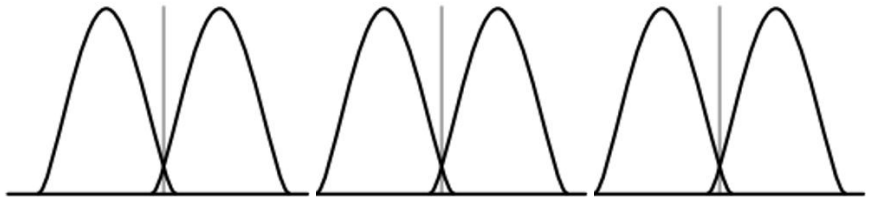
Write Stress Management in Mobile Devices

- Modern apps are becoming write-intensive

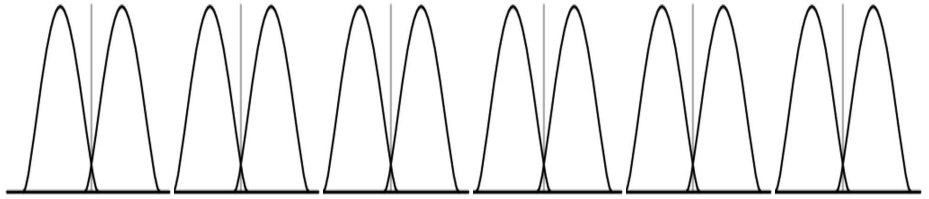


Android smartphones

- Advanced NAND flash: Density ↑ Endurance↓ ↓



3-bit TLC: P/E limit=~3000

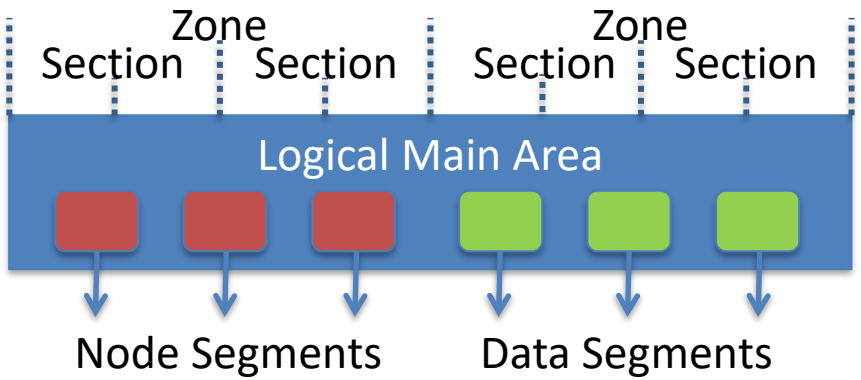


4-bit QLC: P/E limit=~1000

Win-Win: Transparent write stress reduction

Background: F2FS & inode Inline Space

- F2FS - Flash Friendly (Log-Structured) File System

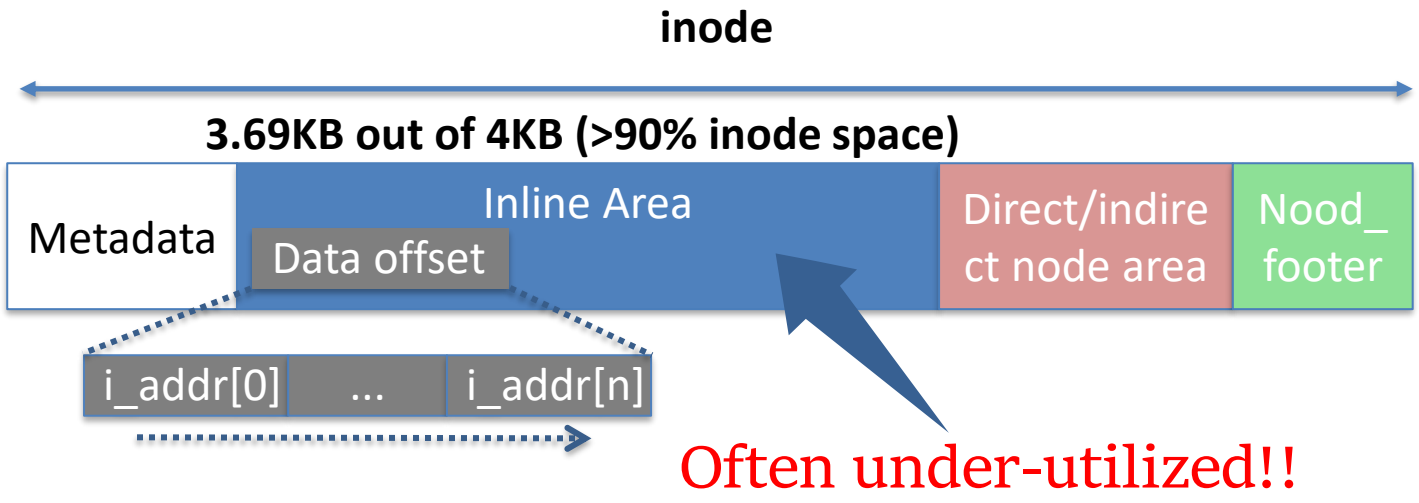


Space format

- Zone
- Section/Segment
- Block

File inode structure

- Inline area - 3.69KB
- Metadata
- Direct/indirect node info
- Node footer

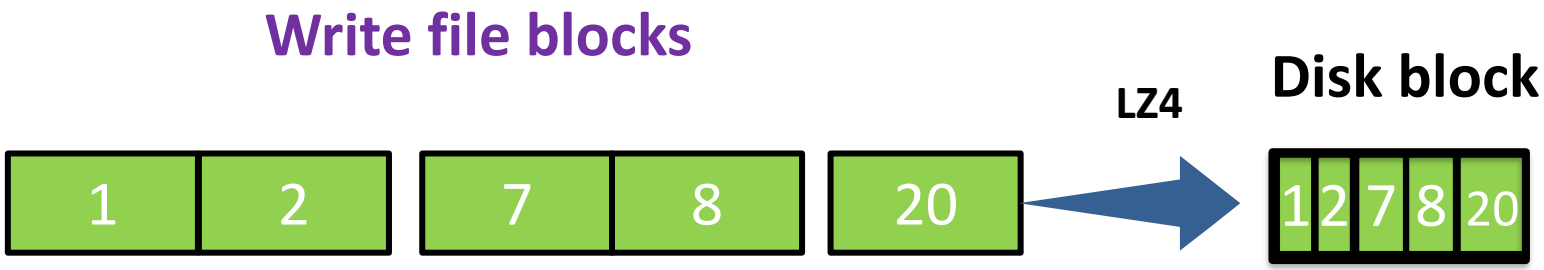


Limitation of Previous Techniques

- Inline files

Only ~25% files can be embedded

- Regular data compression: FPC [Ji FAST'21]



Conventional compression method

- 5 blocks to 1 at most
- High compress/decompress overhead

Less effective on write traffic reduction

Empirical Study on Mobile I/O Workload

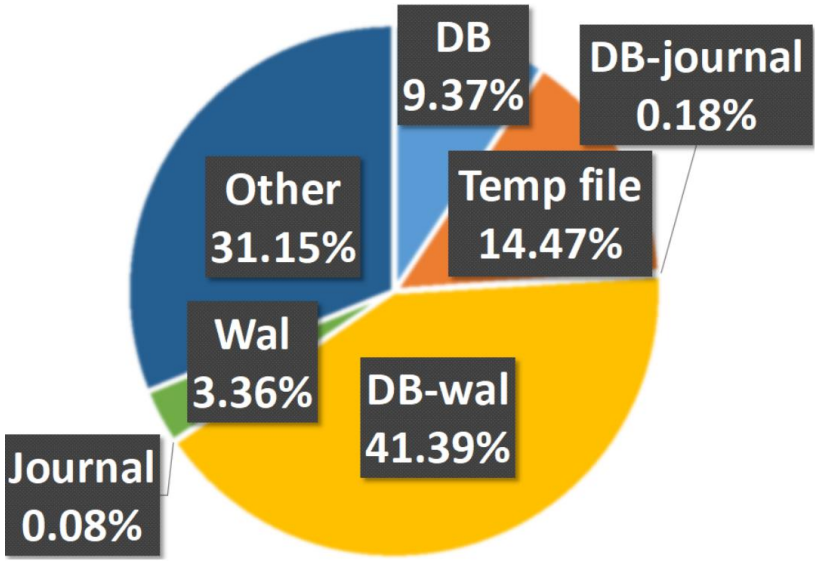
App	Task	Vol	Small	UR	UD	App	Task	Vol	Small	UR	UD
Gmail	Launch app	0.7	100.0%	62.4%	31.2%	Telegram	Receive 1 image	3.5	51.7%	84.4%	1.6%
	Send 1 image	6.3	94.8%	89.6%	24.3%		Receive 10sec voice	4.7	72.1%	86.9%	2.5%
	Send 100 characters	9.0	92.4%	91.5%	25.3%	Twitter	Launch app	9.2	100.0%	61.4%	25.2%
	Receive 1 image	12.3	89.1%	71.1%	30.2%		Post 100 character	12.5	100.0%	92.9%	11.1%
	Receive 100 characters	7.2	98.6%	62.8%	29.0%		Post 1 image	15.8	100.0%	84.5%	8.3%
Polish	Launch app	0.1	100.0%	78.6%	0.9%		Watch news 5min	117.5	99.2%	75.6%	14.4%
	Edit a photo	130.6	99.9%	51.8%	2.7%	Wechat	Launch app	2.0	97.0%	67.1%	14.4%
Spotify	Launch app	4.4	100.0%	93.1%	5.5%		Send 100 character	2.3	94.6%	75.9%	21.7%
	Listen to music 5min	67.4	100.0%	62.3%	11.2%		Receive 100 character	1.8	97.7%	76.5%	22.3%
Telegram	Send 1 image	2.4	52.9%	85.1%	5.2%		Receive 10sec voice	2.8	85.4%	55.3%	9.2%
	Send 10sec voice	3.4	88.1%	92.7%	3.6%		Post 1 image	2.7	92.1%	61.2%	14.7%
	Receive 100 character	4.8	64.4%	95.1%	4.1%	Zoom	Attend 5min meeting	207.4	82.0%	92.5%	13.1%

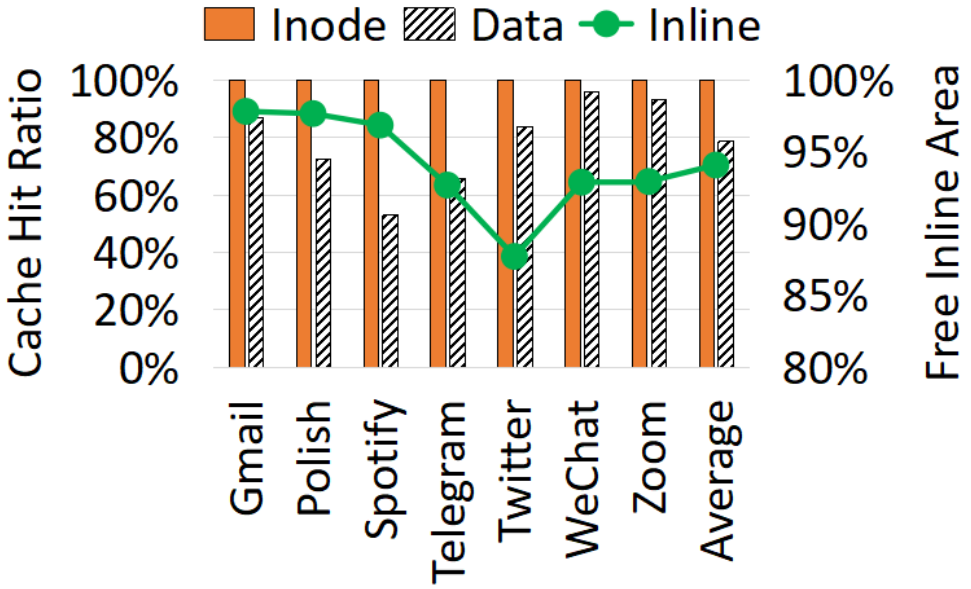
1. 90% files ≤ 3.69 MB (leave free space in inline space)
2. 77.1% writes are updating
3. 13.8% content diff on updating (!)

Delta compression is a good way to go

Empirical Study on Mobile I/O Workload

I/O Writes



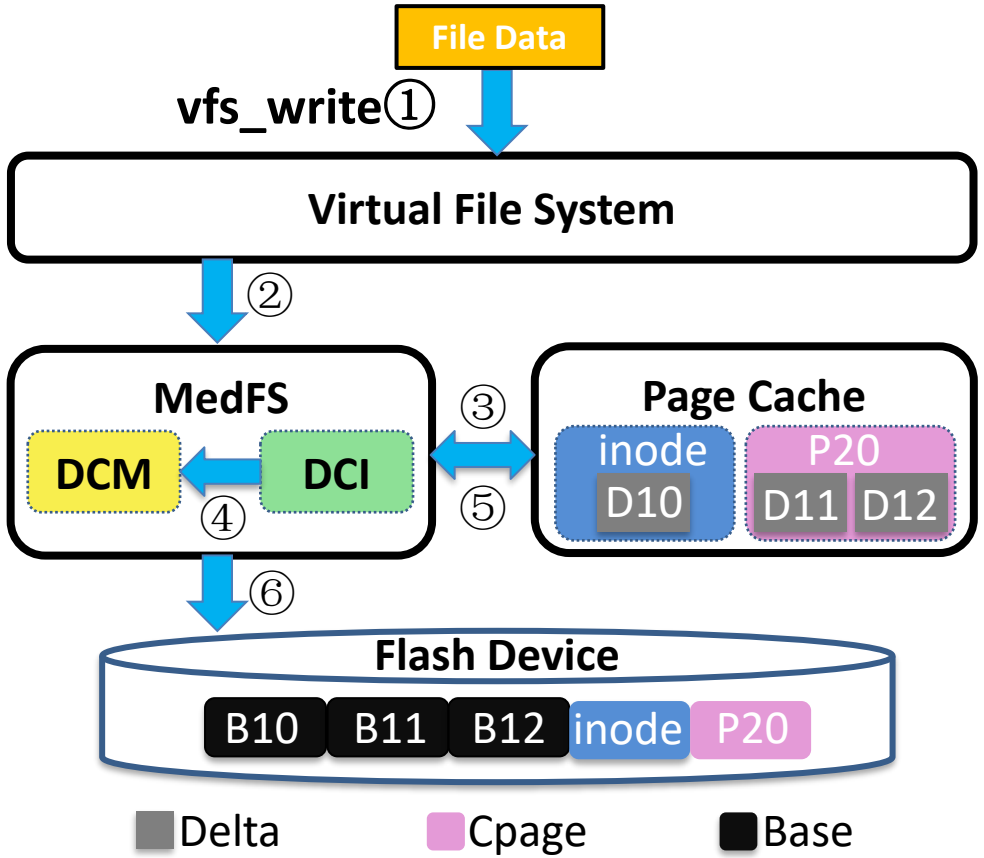


1. SQLite files 54.4%
2. Temp files 14.5%
3. They update with tiny deltas

1. 94% inode inline space is unused → store deltas
2. Inodes cache well (60~80% hits) → multiple deltas until being flushed

Delta-inlining is feasible!

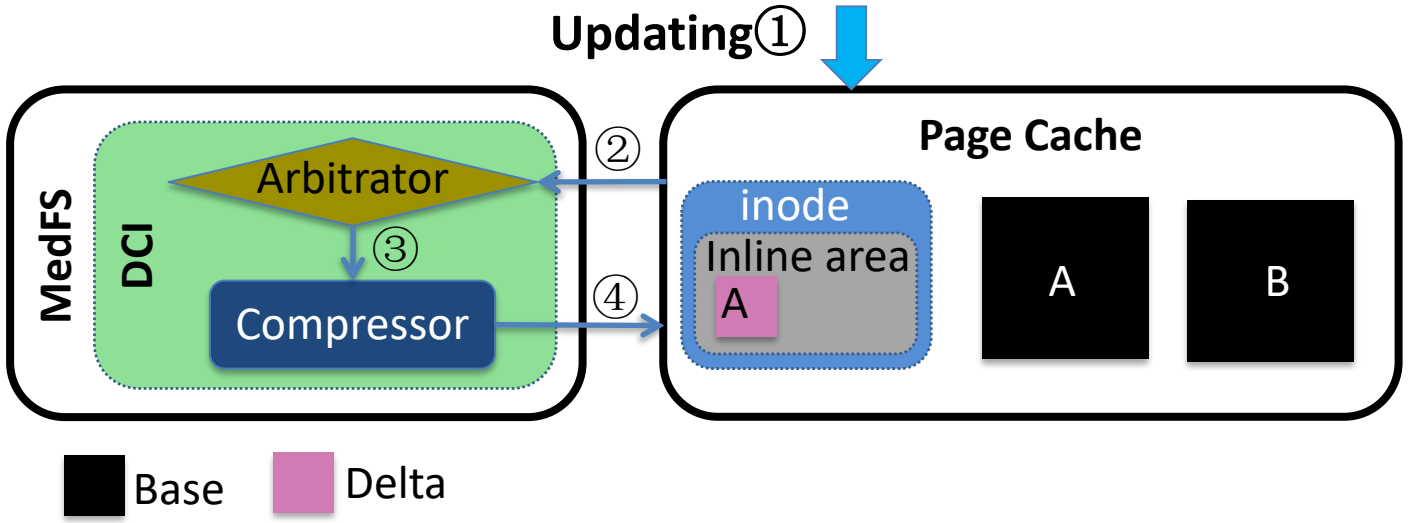
MedFS: Metadata-Enabled Delta Compression for LFS



E.g., D10 is the delta of B10

- **Delta-Chunk Inlining(DCI)**
  - Storing deltas in inodes
  - Reducing block write count
- **Delta-Chunk Maintenance(DCM)**
  - Overflowing deltas to data blocks
  - Going beyond the inline size limit

Delta-Chunk Inlining (DCI)



Delta = compressed XOR

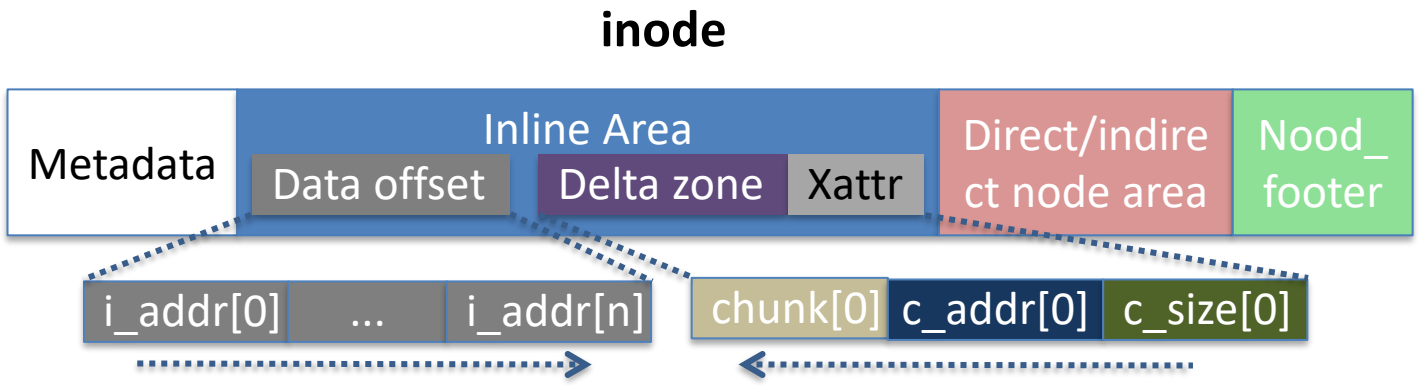
000011100000011110

000010000000000000

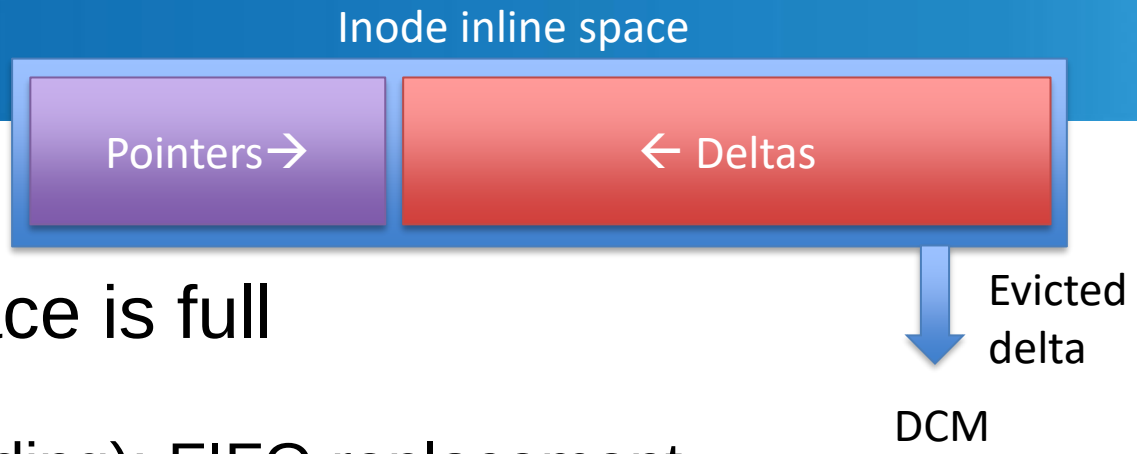
000001100000000000

Run-length coding or LZO

A base block can have only 1 delta (no delta on delta)



Delta-Chunk Inlining (DCI)



- Delta is replaced when inline space is full

  1. Adding a new pointer (file appending): FIFO replacement

  2. Updating w/o an existing delta: $Benefit = \frac{RD}{\alpha} \times \beta$ (write reduction)

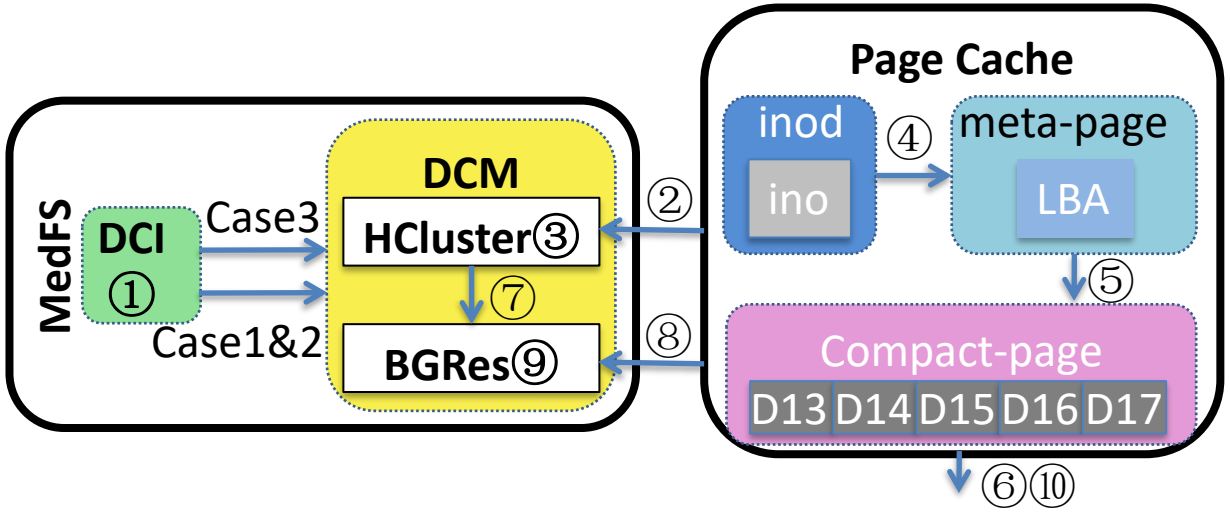
     $Overhead = HR \times \lambda + (1 - HR) \times \delta + \varepsilon + \beta$ (decompress cost)

     Replace a delta if benefit>overhead

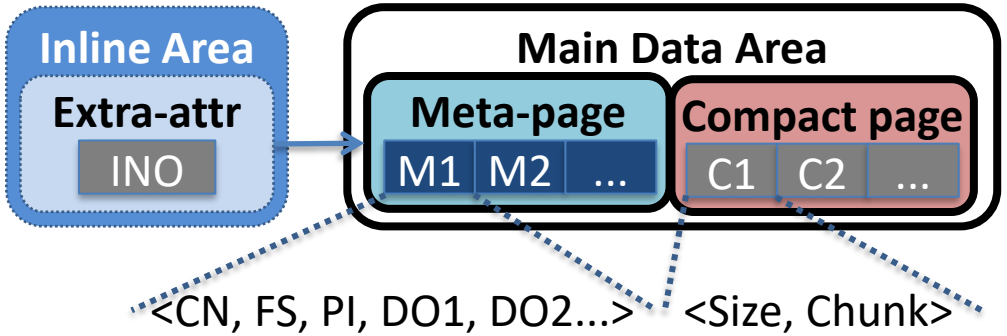
  3. Updating w/ an existing delta: if new delta < old delta, replace

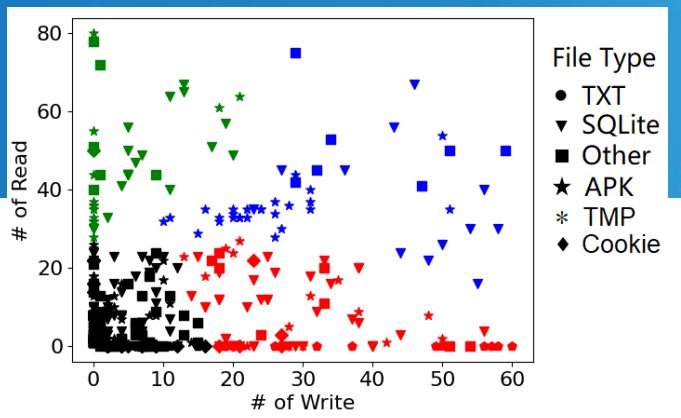
- Evicted delta is forwarded to DCM

Delta-Chunk Maintenance (DCM)



Meta page: metadata of deltas
Compact page: deltas

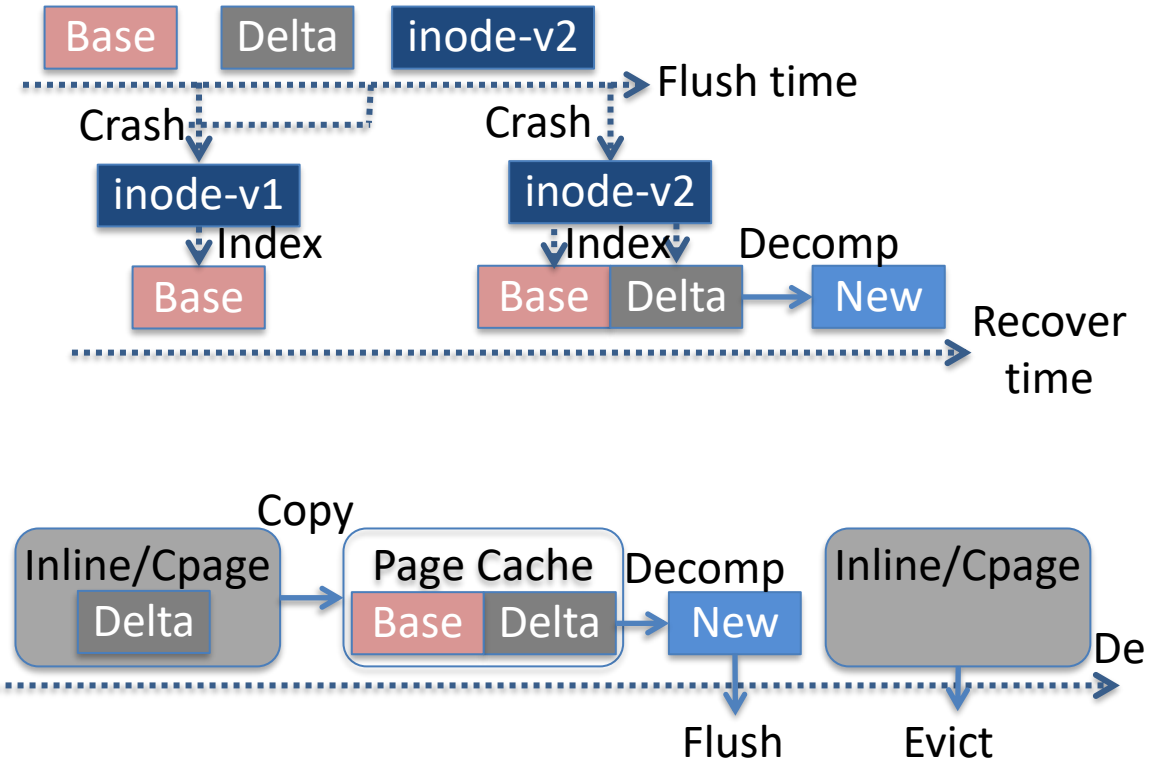




- **DCM accepts evicted deltas**
- **HCluster (clustering)**
  - Identify **read-cold-write-hot** file
  - If yes, store deltas in meta-page and compact page
  - If not, decompress
- **Delta-Chunk Maintenance(DCM)**
  - A kernel thread
  - Decompress a file if no longer RCWH

Consistency / Recovery

- MedFS enforces two write orderings



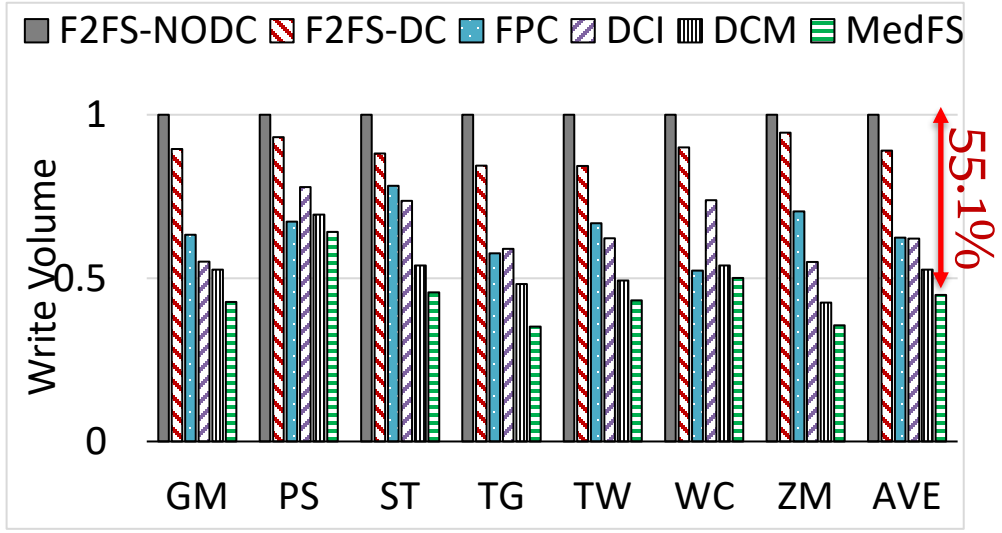
- **Base ≺ delta blocks ≺ inodes**
  - A base is always flushed before its associated delta
  - To avoid dangling delta
- **Decompressed block ≺ updated inode**
  - To avoid lost update
- **Recovery is based on F2FS**

Experimental Setup

- Implementation on Oneplus 8T
  - Snapdragon 865, 8GB RAM, and 128GB of UFS3.1
  - Android14 with Linux kernel 4.19
  - File System: ***F2FS***
- Six related methods are evaluated
  - Original F2FS (***F2FS-NODC***), F2FS with conventional compression (*F2FS-DC*), ***FPC*** *[FAST'21],* ***DCI-only*** *(DCI),* ***DCM-only*** *(DCM), DCI+DCM (MedFS)*
- The evaluation is based on popular mobile apps
  - Apps: Gmail, Polish, Spotify, Telegram, Twitter, Wechat, Zoom
  - Metrics: write Stress, user perceived latency, I/O latency, etc.

Experimental Results

◆ Results of Write Stress Release



$$LIFETIME = \frac{PEC \times (1 + OP)}{365 \times DWPD \times WA \times R_{COMPRESS}}$$

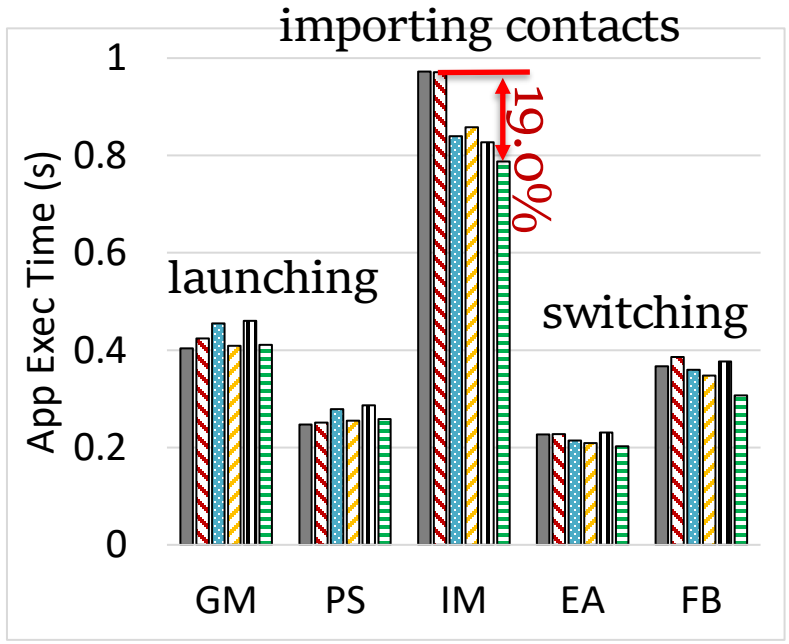
→ +122.7% extra lifespan

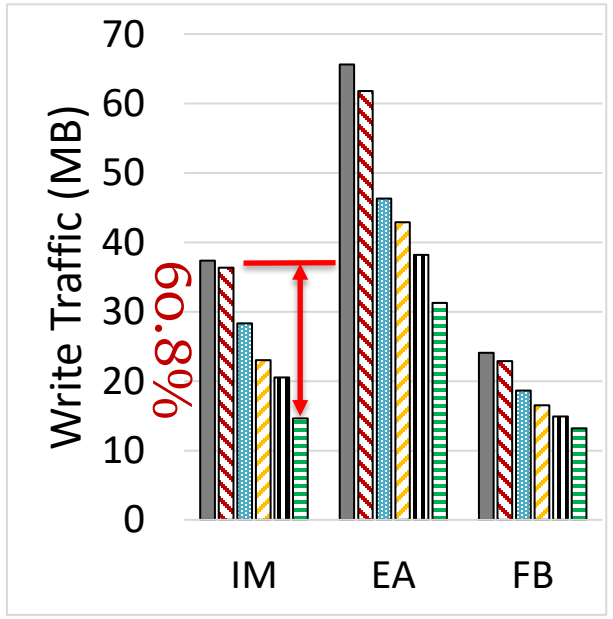
Delta compression works great on tiny changes

Improving flash storage lifetime significantly

Experimental Results

◆ User Perceived Latency



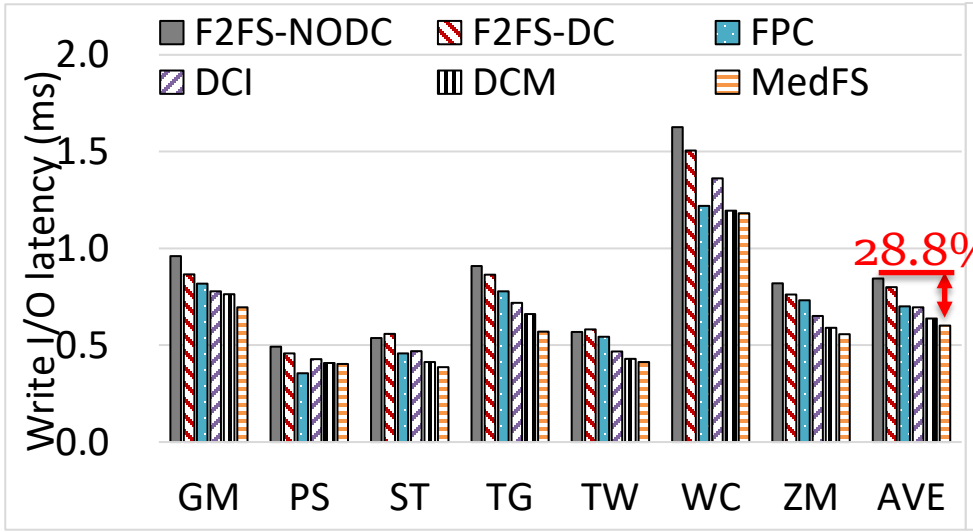


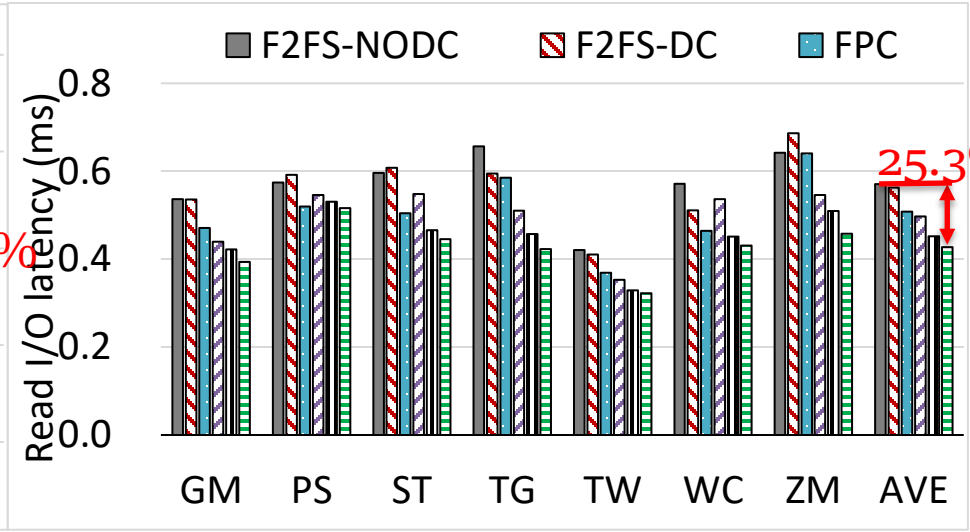
Improving UPL for write-dominant scenarios

Minimally impacting on read-intensive scenarios

Experimental Results

◆ I/O latencies





Delta-embedding Inode hits (caches) well → write I/O latency ↓
→ scheduler congestion ↓ → read I/O latency ↓

Conclusion

- Mobile apps make tiny changes to SQLite/temp files through synchronous writing
  - Amplifying write & reducing flash lifespan
- Delta compression works great in handling tiny changes
  - Embedding tiny deltas to inode for free writing (DCI)
  - Evicting deltas to delta blocks (DCM)
- We focus on change, conventional compression is on data pattern
  - More effective than compression

Thank You!!

MedFS: Pursuing Low Update Overhead via Metadata-Enabled Delta Compression for Log-structured File System on Mobile Device