D4C: Leveraging Delta Encodings for Faster and Lighter Container Image Updating

FOSDEM 2025 (1 Feb. Dev room - Containers)

Naoki Matsumoto (Kyoto University, Japan)

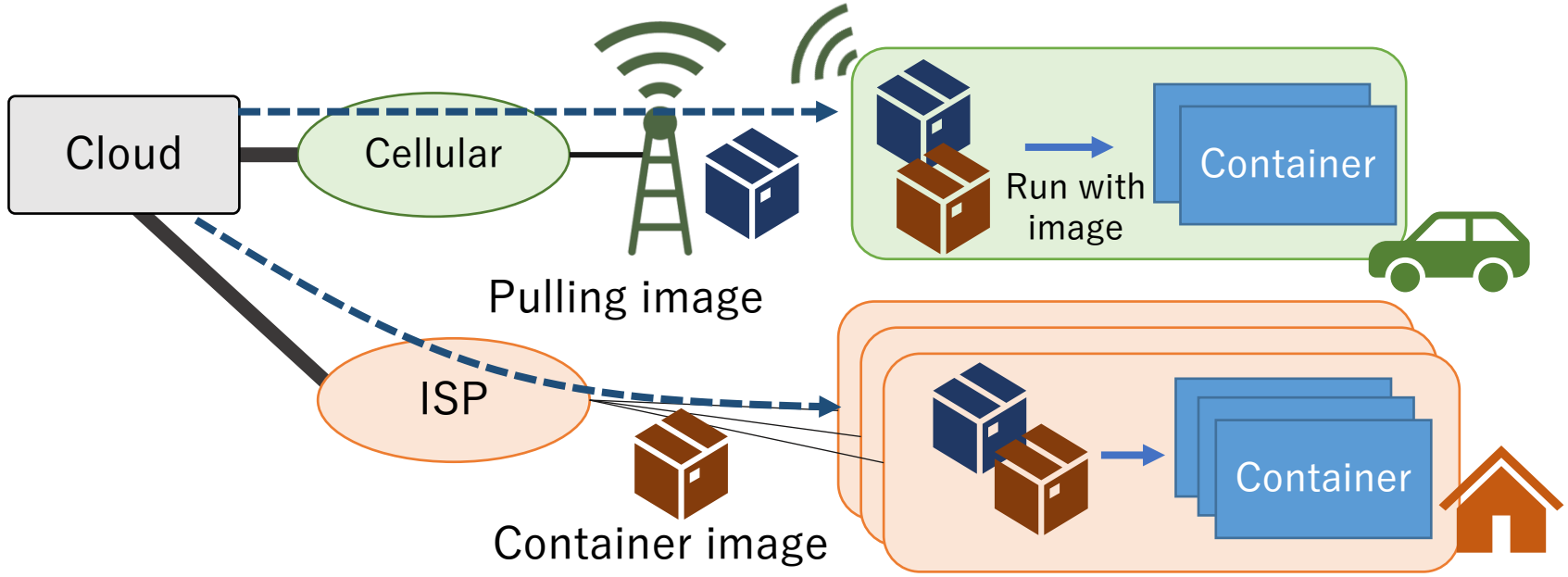
京都大学

Background

Increasing container use in **network-resource restricted environment**

- Bandwidth is low (e.g., Cellular :50~300Mbps[1])

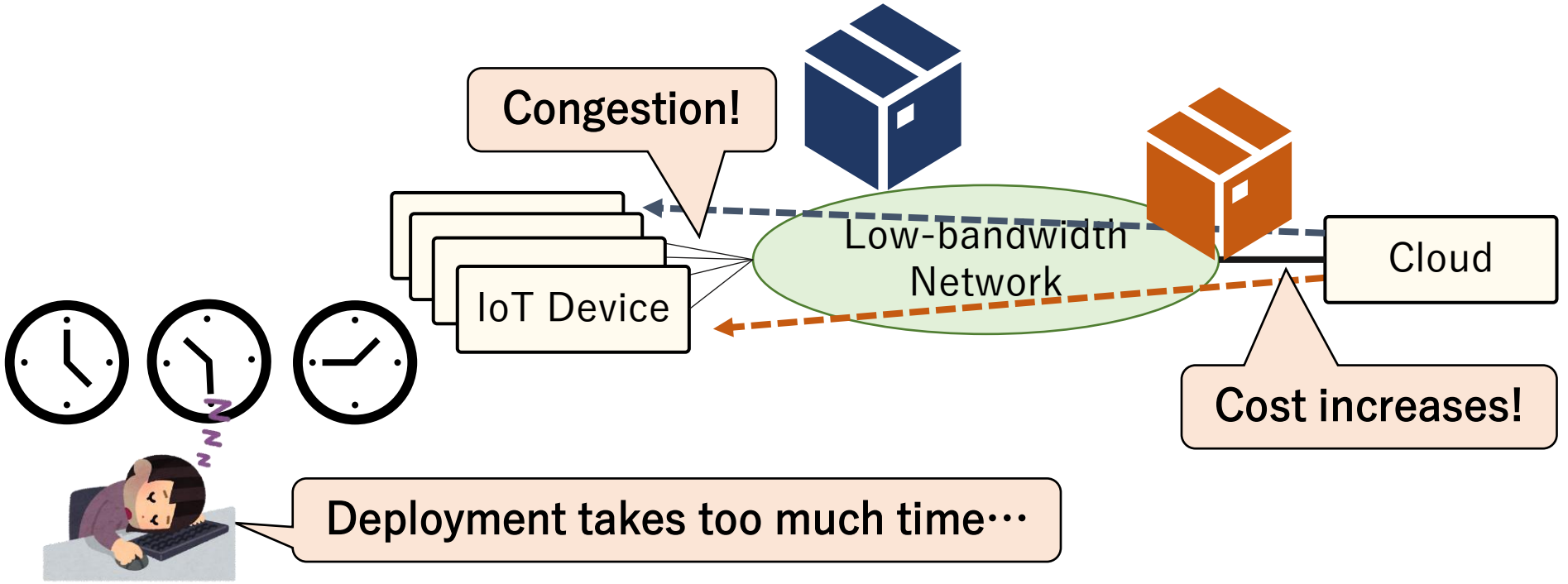
To start or update containers, users download and expand container images (pull)



[1]Mobile access bandwidth in practice: measurement, analysis, and implications(Xinlei Yang et al., 2022)

Problems in Container Image Updating

Large update data cause problems



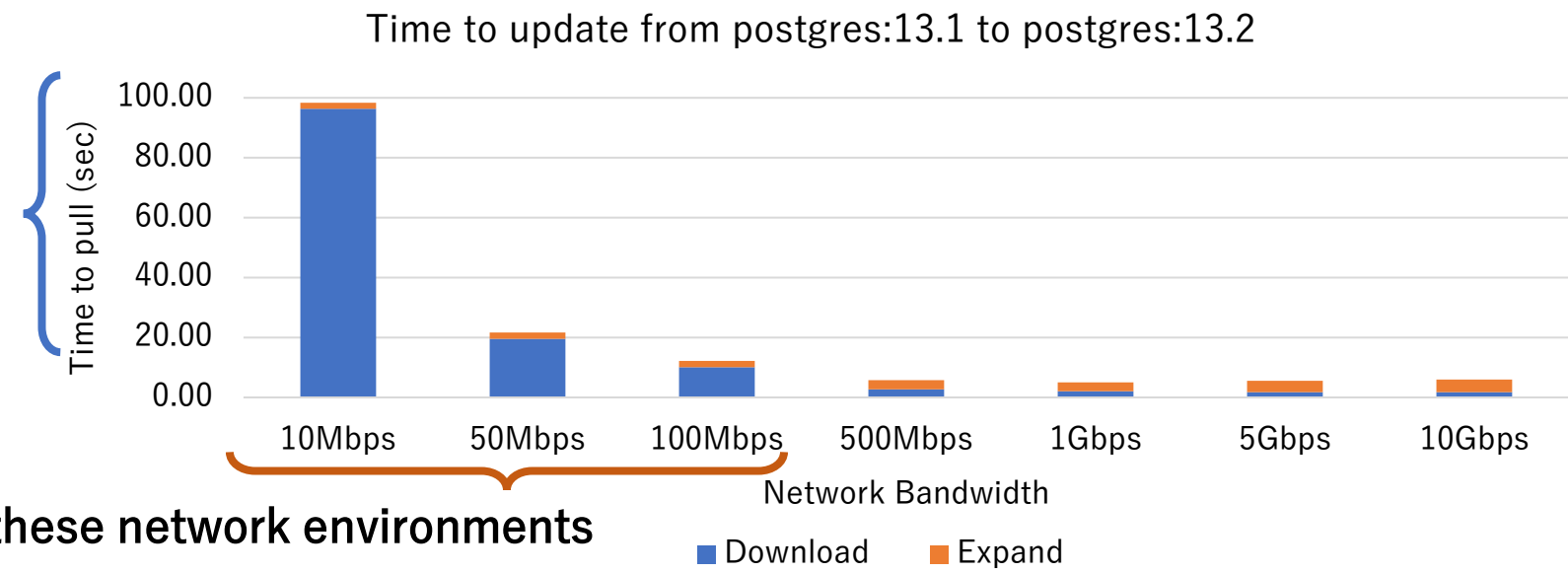
Lightweight and Fast Updating is Needed!

Current Container Image Updating

Current container runtimes (e.g., containerd) provides layer-based image

Layer-based images has a limitation to provide efficient update[2]

Download time is dominant



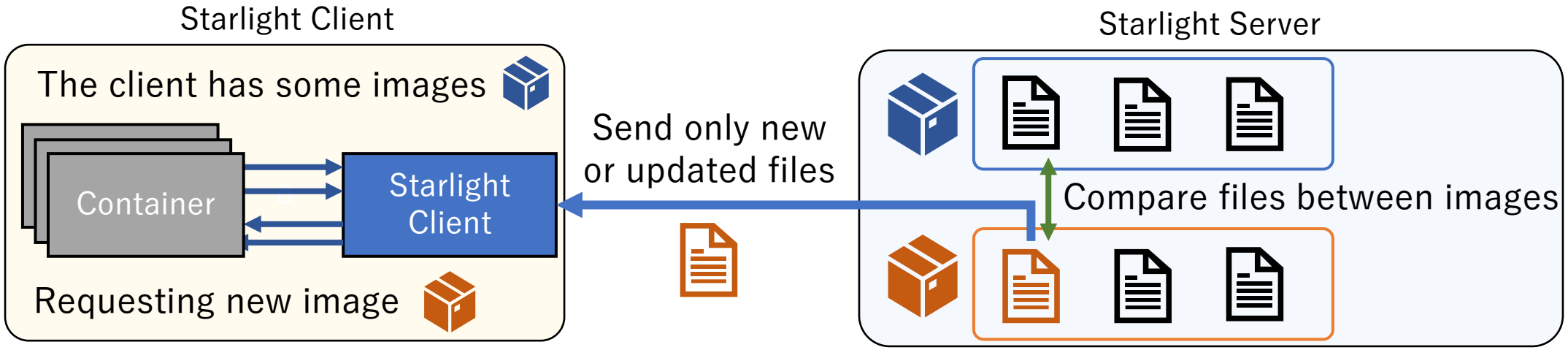
We assume these network environments

[2] Starlight: Fast Container Provisioning on the Edge and over the WAN (Jun Lin Chen et al., 2022)

Related Works: File-oriented deduplication

Starlight[2], zstd:chunked

- Pulling <u>only new or updated files</u>
- zstd:chunked implements the same approach



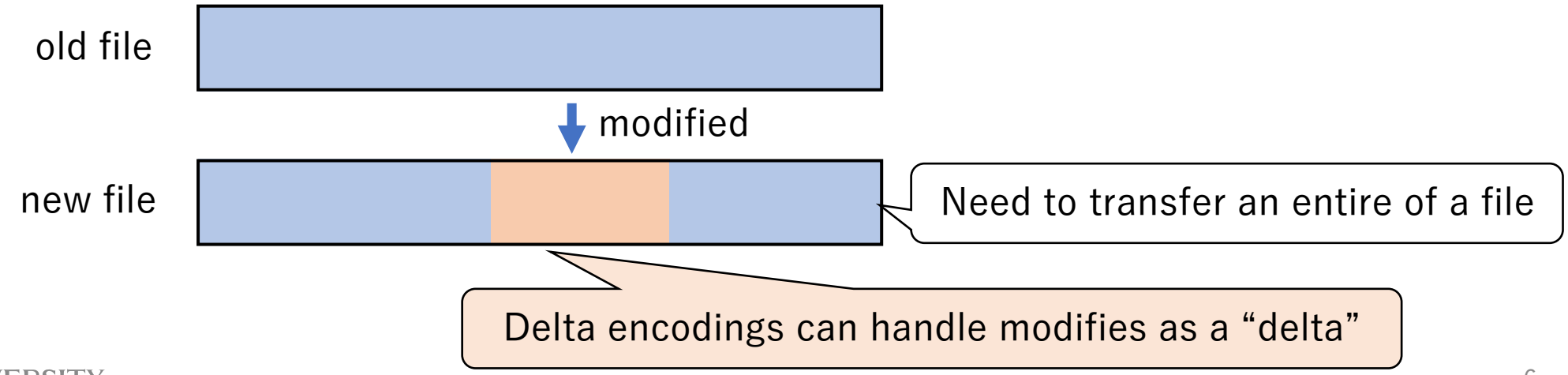
[2] Starlight: Fast Container Provisioning on the Edge and over the WAN (Jun Lin Chen et al., 2022)

Problems in Related Works

These works rest a room to reduce data size for update

File-oriented deduplication

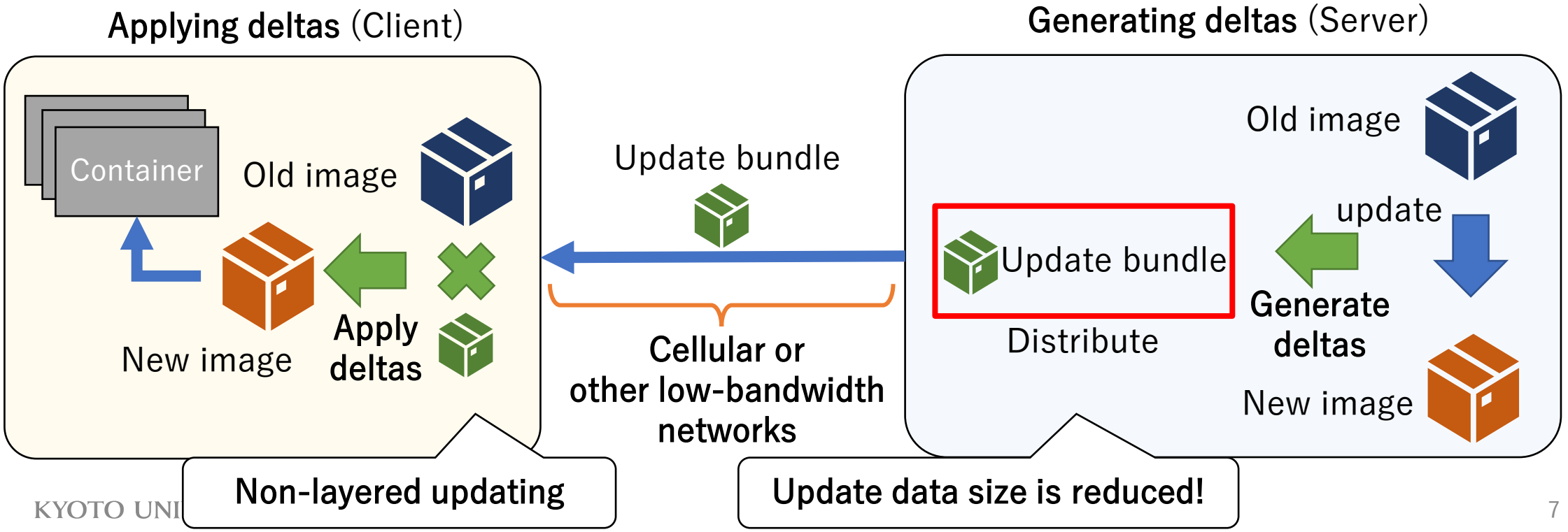
- Cannot handle **partial modifications** on files efficiently
- Most of the content in some execs and shared libs are not updated



An Approach of D4C

Reducing data to update images using delta encodings

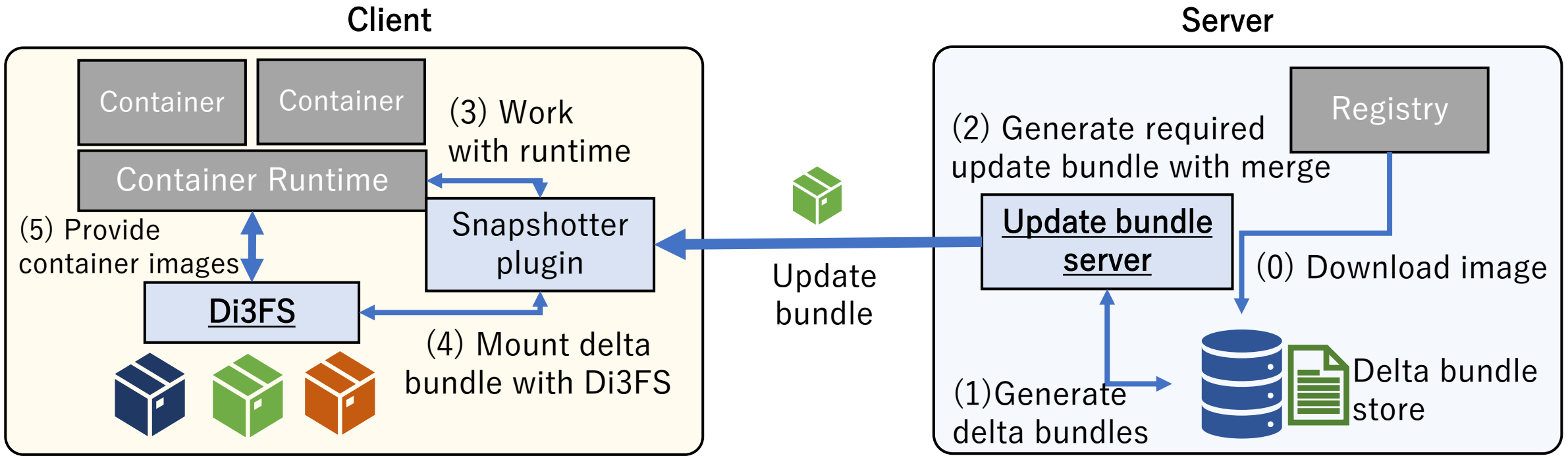
- Transferring only required partial data to update



Overview of D4C

D4C uses **merge strategy** for container image updating

- A server generates and merges deltas, and a client applies deltas to old images

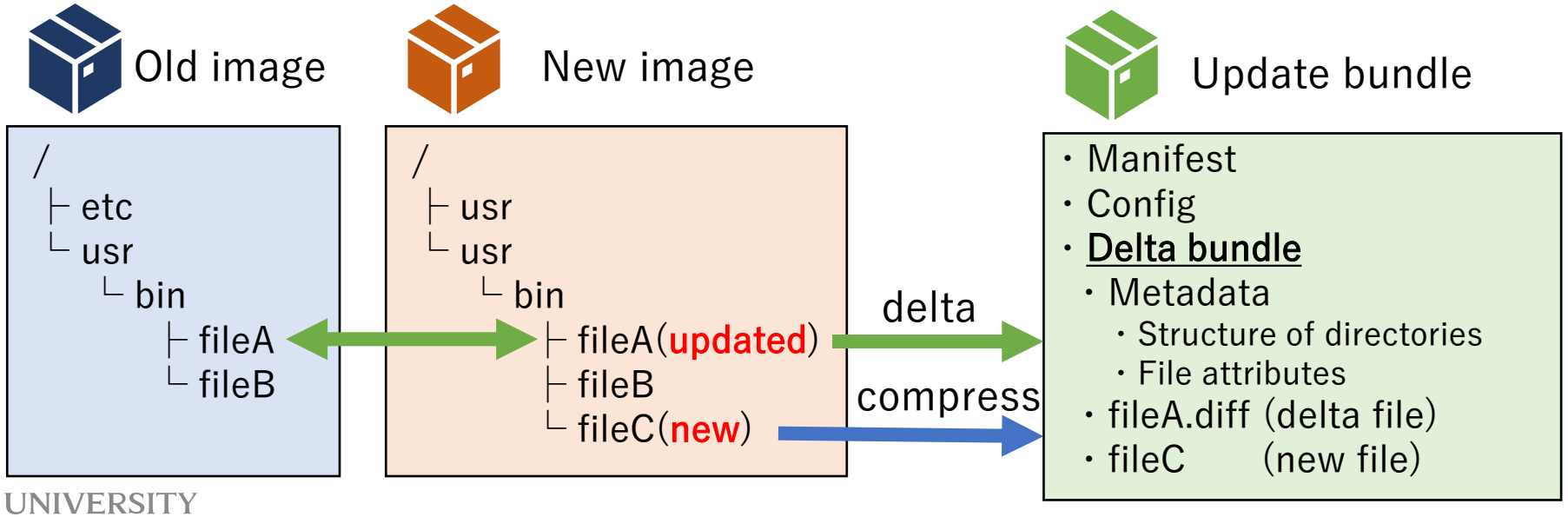


Delta Generation

Generating deltas for each file and packing them as delta bundle

- Delta encoding generates delta files for updated files
- New files are compressed

Manifest and Config for container are packed as an update bundle



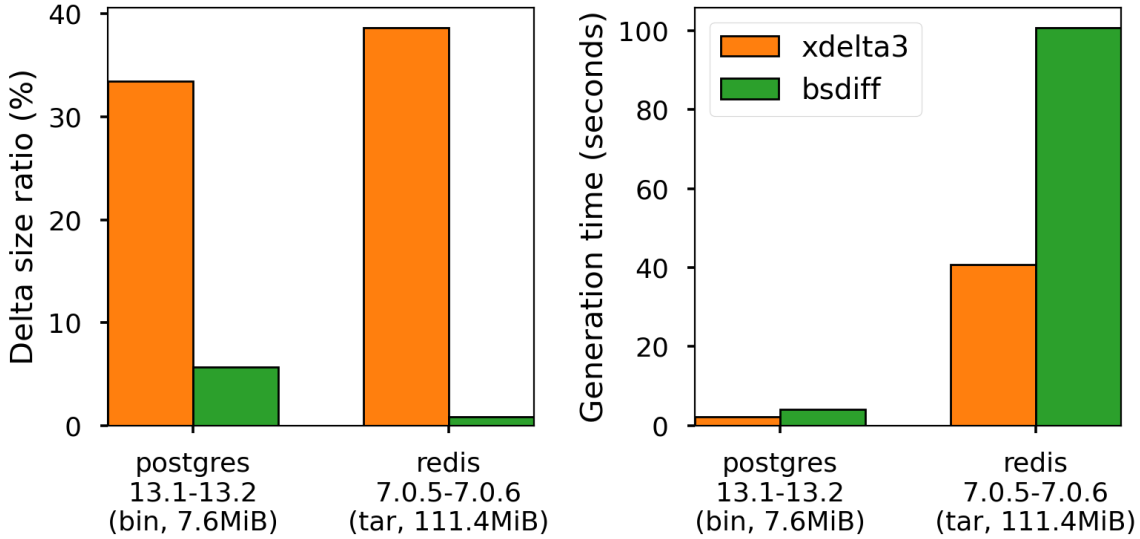
Issues to Utilize Delta Encodings

Generating deltas takes much time

- Better compression requires longer time

Longer generation increases overall updating time

→**How to provide requested deltas quickly?**

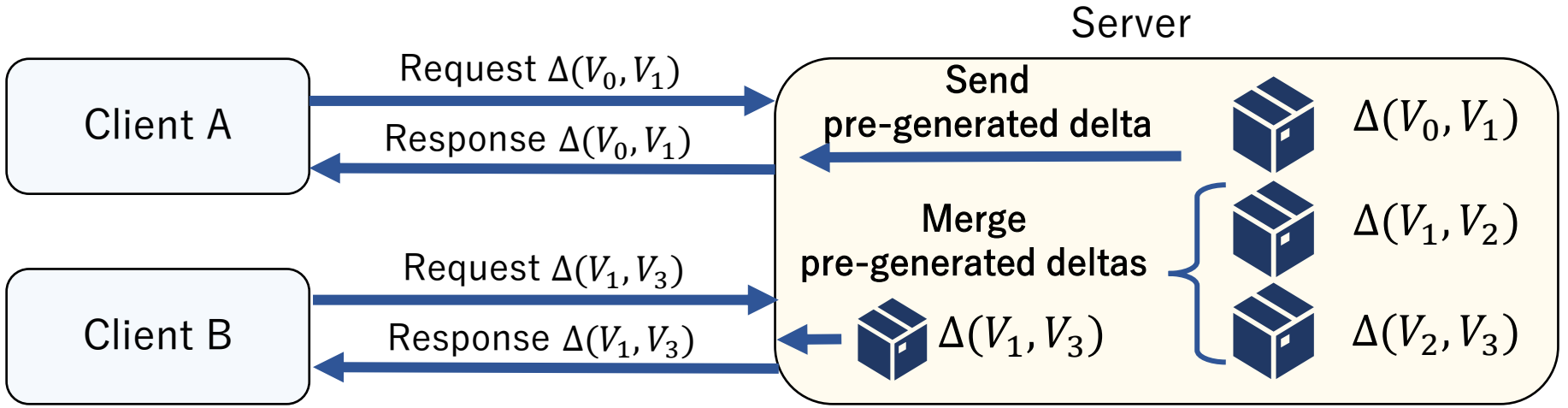


Strategy for Fast Delta Generation

D4C employs the approach to **utilize pre-generated deltas and merging**

💡 Merging deltas **does not take much time** than generating them from scratch

- **Generating deltas for (V_i, V_{i+1}) in advance and merging them when requested**



Supported Delta Encoding Algorithms

D4C treats delta encodings as a “Plugin” with simple API

- **Generate**(base, updated) → delta
- **Apply**(base, delta) → updated
- **Merge**(deltaA, deltaB) → deltaC

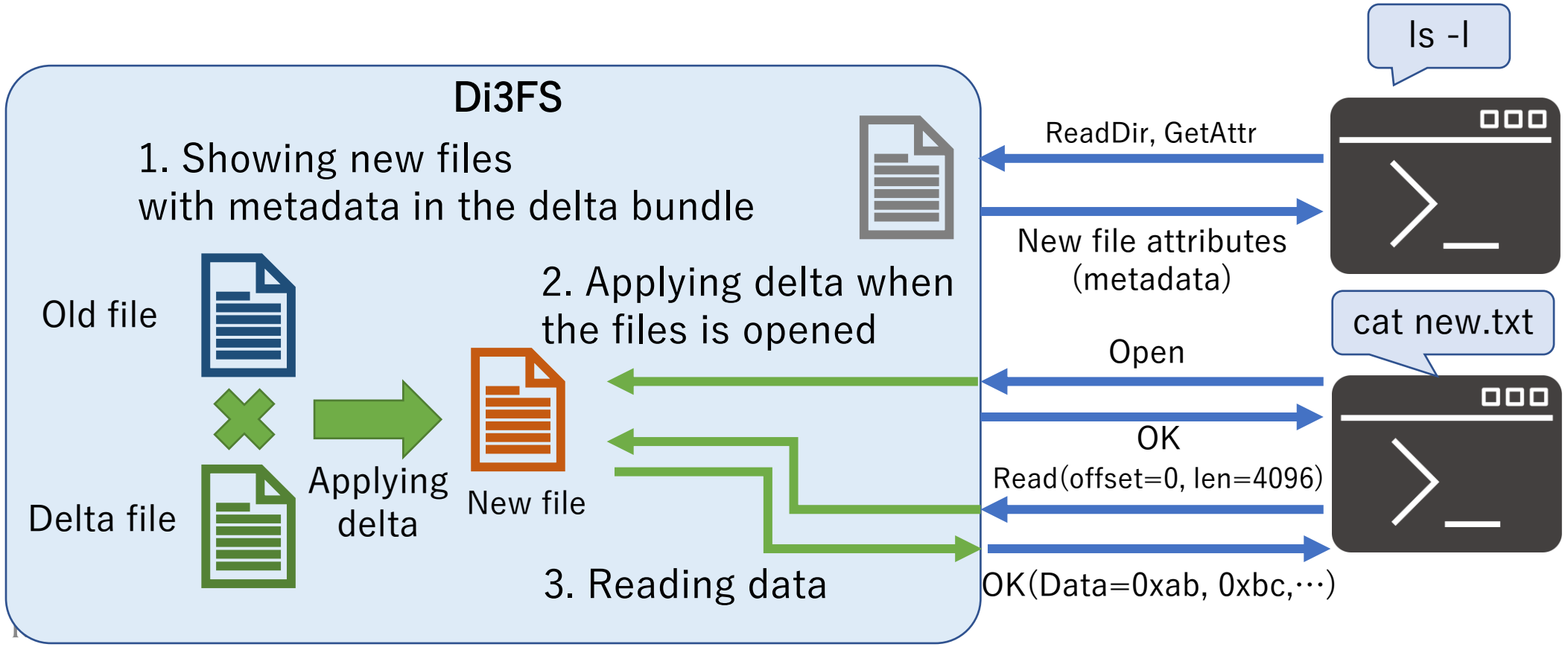
D4C has 2 plugins based on “**bsdiff**” and “**xdelta3**”

- bsdiff does not have “Merge”. D4C provides newly implemented “Merge”
- xdelta3 provides “Merge” only via CLI

Lazy Delta Applying: Di3FS

Applying deltas on-demand when the file is opened → No need to apply all deltas

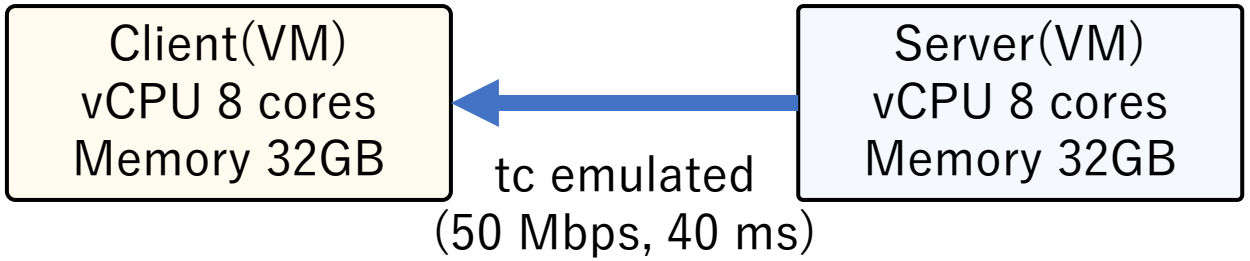
same approach with lazy-pulling



Implementation and Evaluation

Environment: Slow cellular network

- Parameters are Throughput: 50 Mbps, Latency(RTT): 40 ms [1][4]



Used Image	Tag (size)
postgres	13.1(109.44MB), 13.2(109.51MB), 13.3(109.62MB)
redis	7.0.5(40.43MB), 7.0.6(40.44MB), 7.0.7(40.44MB)
nginx	1.23.1(54.14MB), 1.23.2(54.19MB), 1.23.3(54.25MB)
pytorch	cuda12.1-cudnn8-runtime- 2.2.0(3.41GB), 2.2.1(3.41GB), 2.2.2(3.73GB)

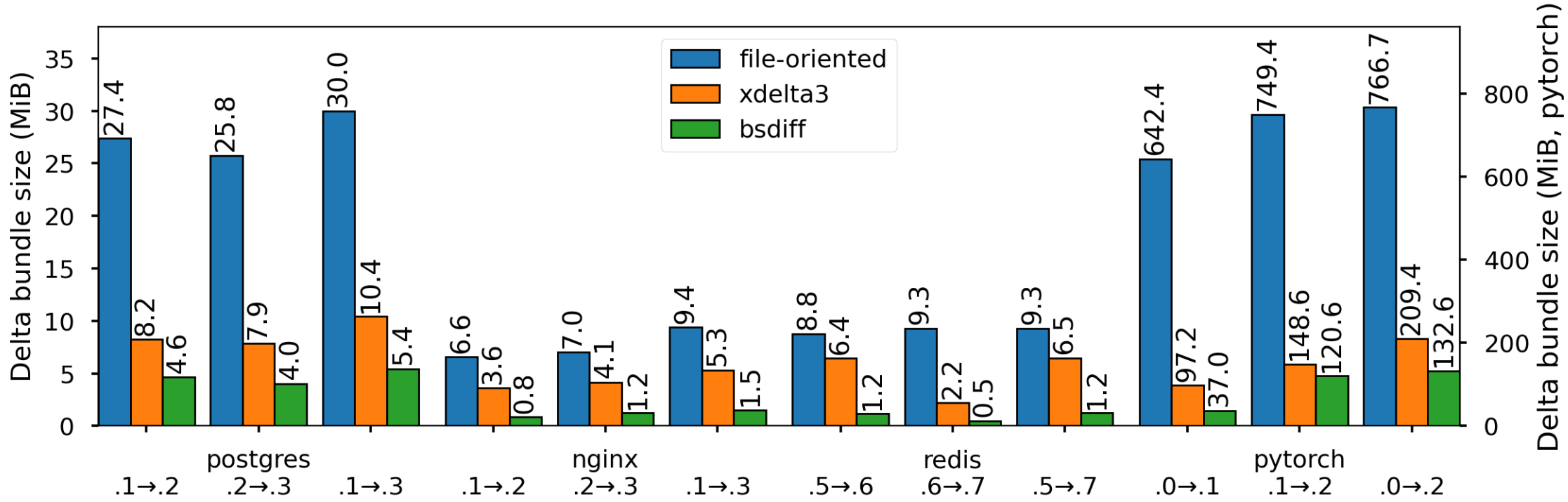
[1]Mobile access bandwidth in practice: measurement, analysis, and implications(Xinlei Yang et al., 2022)
[4] Revisiting the Arguments for Edge Computing Research(Blesson Varghese, et al., 2021)

Data Size Reduction to Update Images

Compared delta size reduction with file-oriented deduplication approach

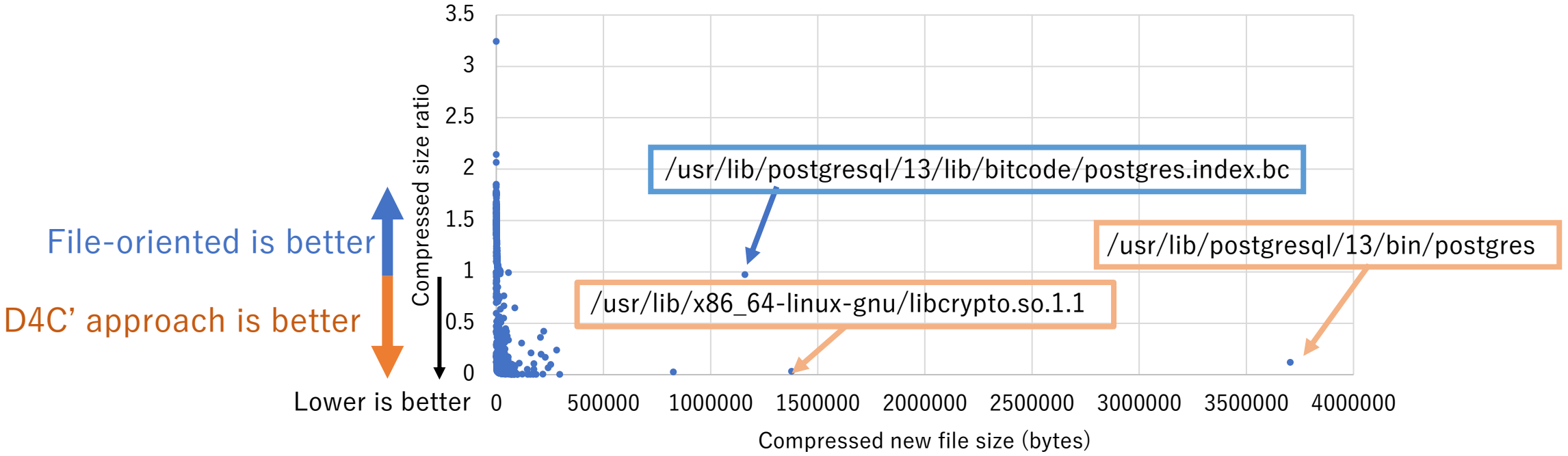
→ D4C provides **deltas only 5〜40%** size compared to file-oriented deltas

＝ **20x compression** at most!



Breakdown of Size Reduction

- Huge size reductions were seen in **executables and shared libs**
- Deltas for compressed or bit-encoded files were inflated

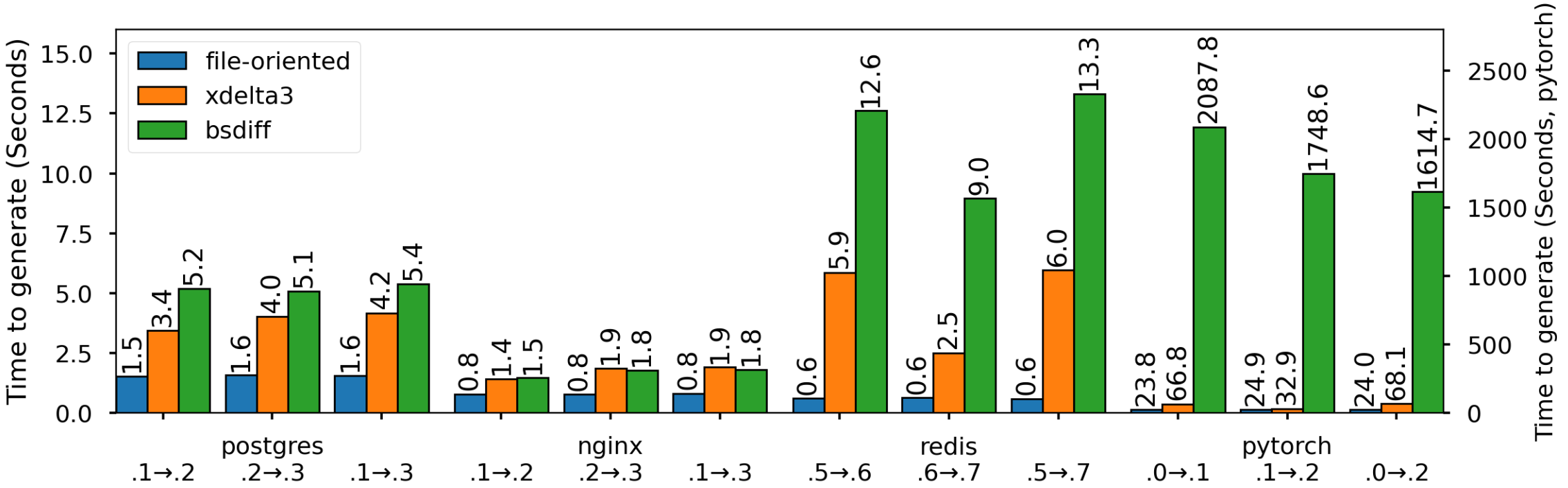


Deltas between postgres:13.1 and postgres:13.2 with bsdiff

Time to Generate Deltas

Time to generate deltas increased compared to file-oriented approach

- bsdiff took much time due to large files (over 100MiB files)
- xdelta3 did faster than bsdiff, but it still takes time compared to file-oriented

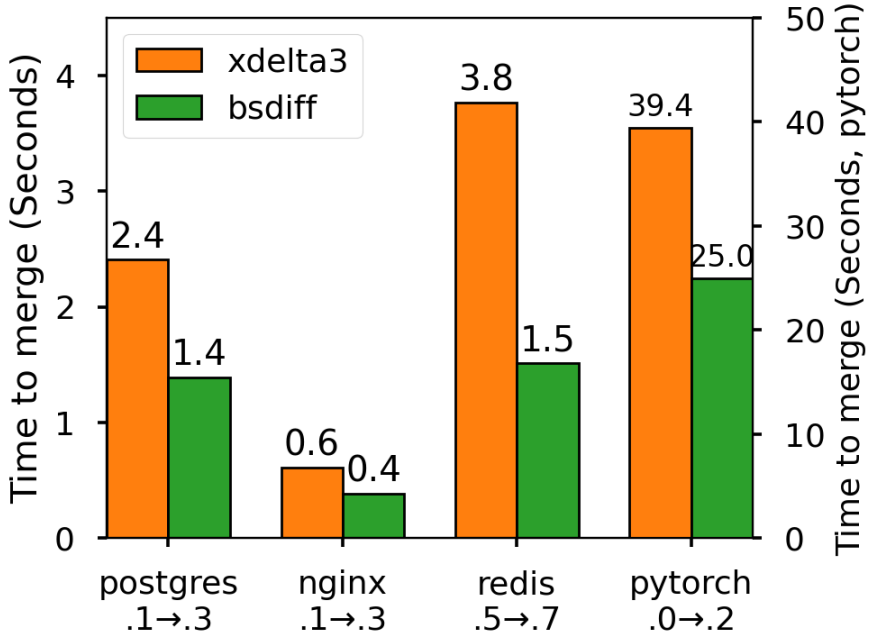


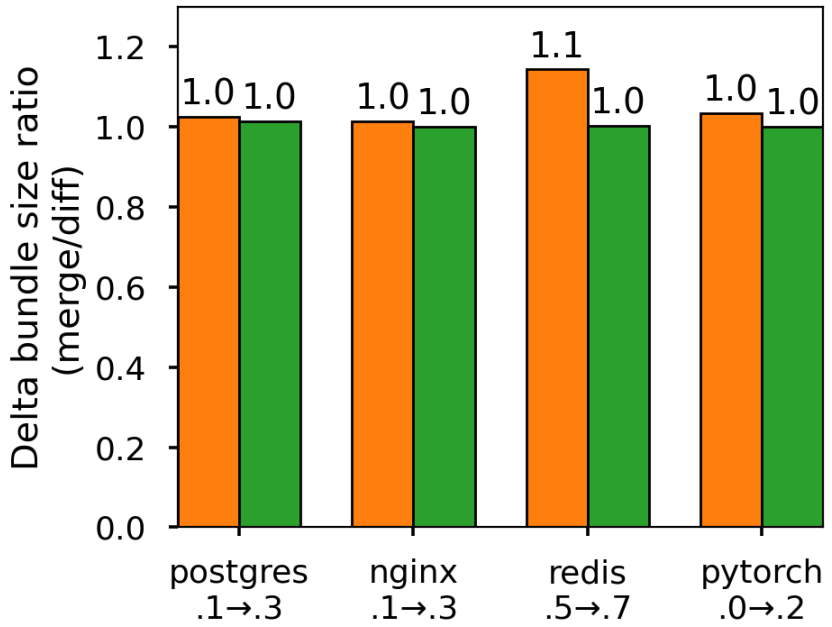
Time to Merge Deltas

Evaluated the performance to merge .X→.X+1 and .X+1→.X+2 deltas

Merge provides deltas fast with less size inflation

- **Merge ran 65x faster than generating from scratch for pytorch with bsdiff**
- Merged deltas had almost same size with generated ones



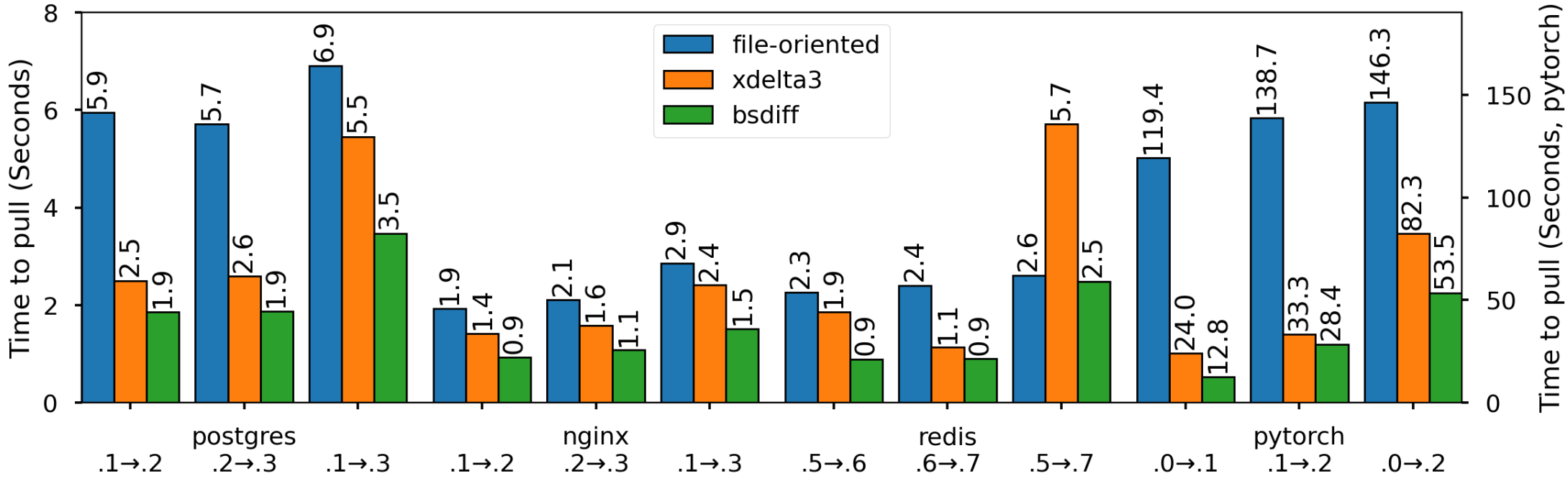


Time to Update Container Images

Measured in 50 Mbps limited client-server network environment

- As for case .X → .X+2, the server merges deltas on-demand

→ D4C provides updates 10x faster update at most



Performance Degradation on Applications

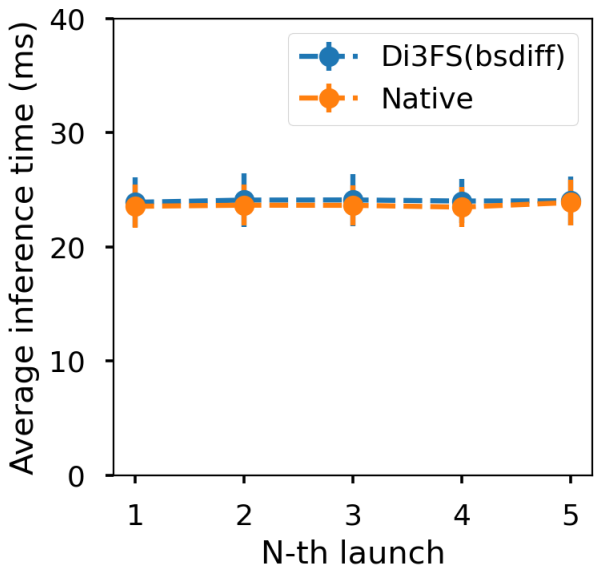
Evaluated updated images(postgres, pytorch)

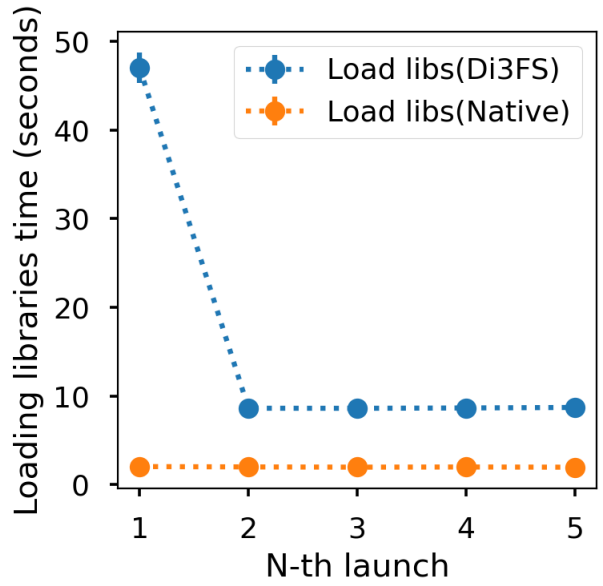
No performance degradation were not seen in benchmarks

- Di3FS caches delta-applied files on a storage when a file is opened for the first time
- First open (e.g. library loading) will take time, but the performance is not affected

pgbench result

approach	Time per transaction (ms)	Transactions per second
Di3FS	11.549 ± 0.722	869.453 ± 57.484
Native FS	11.540 ± 0.843	871.386 ± 65.541





Next Step

More and more works are left

Current implementation is just a PoC and lacks many features

- Sophisticated CLI tools, Server's WebUI, etc..
- **How to decide the deltas generated in advance?**

Seeking a combination with ztsd:chunked

- Providing updated chunks with delta encodings will be beneficial
- **How to choose the base and updated chunks to generate deltas?**

Summary and Questions?

Objective: Reducing data size and time to update container images

Solution: Utilizing delta encodings

Evaluation: D4C provides 20x compression compared to file-oriented

- Huge reduction in executable binaries and shared libraries
- Performance degradation is little excepting some cases

Next step: Implementing more and more

D4C is available at
https://github.com/naoki9911/d4c