DupHunter: Flexible High-Performance Deduplication for Docker Registries

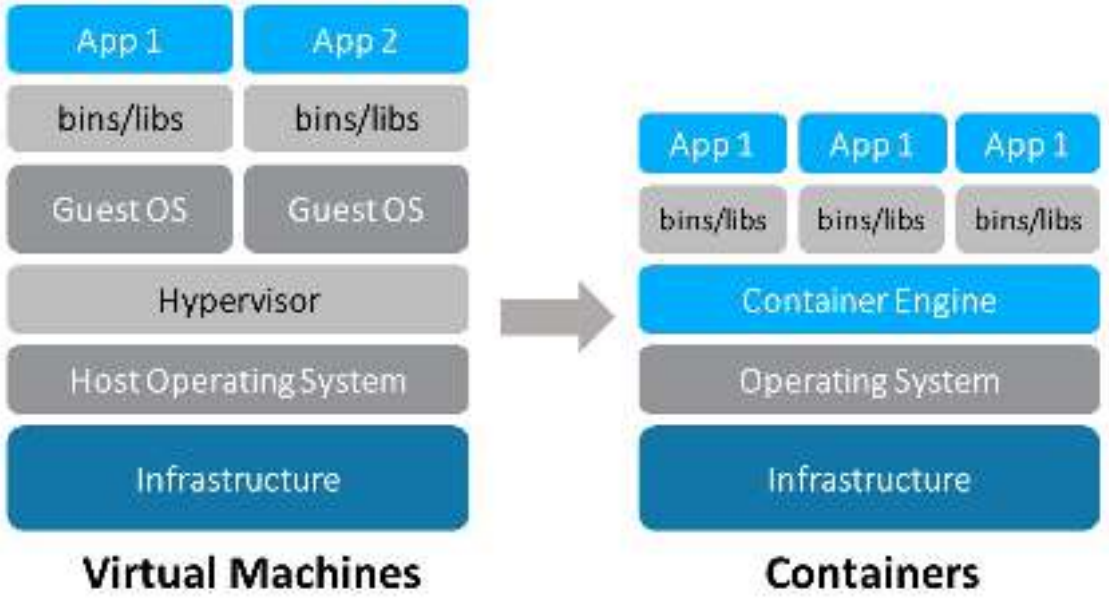
Nannan Zhao[1], Hadeel Albahar[1], Subil Abraham[1], Keren Chen[1], Vasily Tarasov[2], Dimitrios Skourtis[2], Lukas Rupprecht[2], Ali Anwar[2], and Ali R. Butt[1]
[1]Virginia Tech [2]IBM Research—Almaden

ATC 2020

Background

➢ Docker

- Process-level virtualization
- Fast deployment & near-metal performance
- Used widely in production



➢ Docker registry

- Act like GitHub
- The most official: Docker Hub, which stores more than ***15PB*** of images[1]
- Grow by about 1,500 new public repositories daily[Anwar, FAST'18]

Background

➢ Docker architecture overview

- Layer in images are read-only (static)
- Container is a run-time entity
- Image manifest records layer information

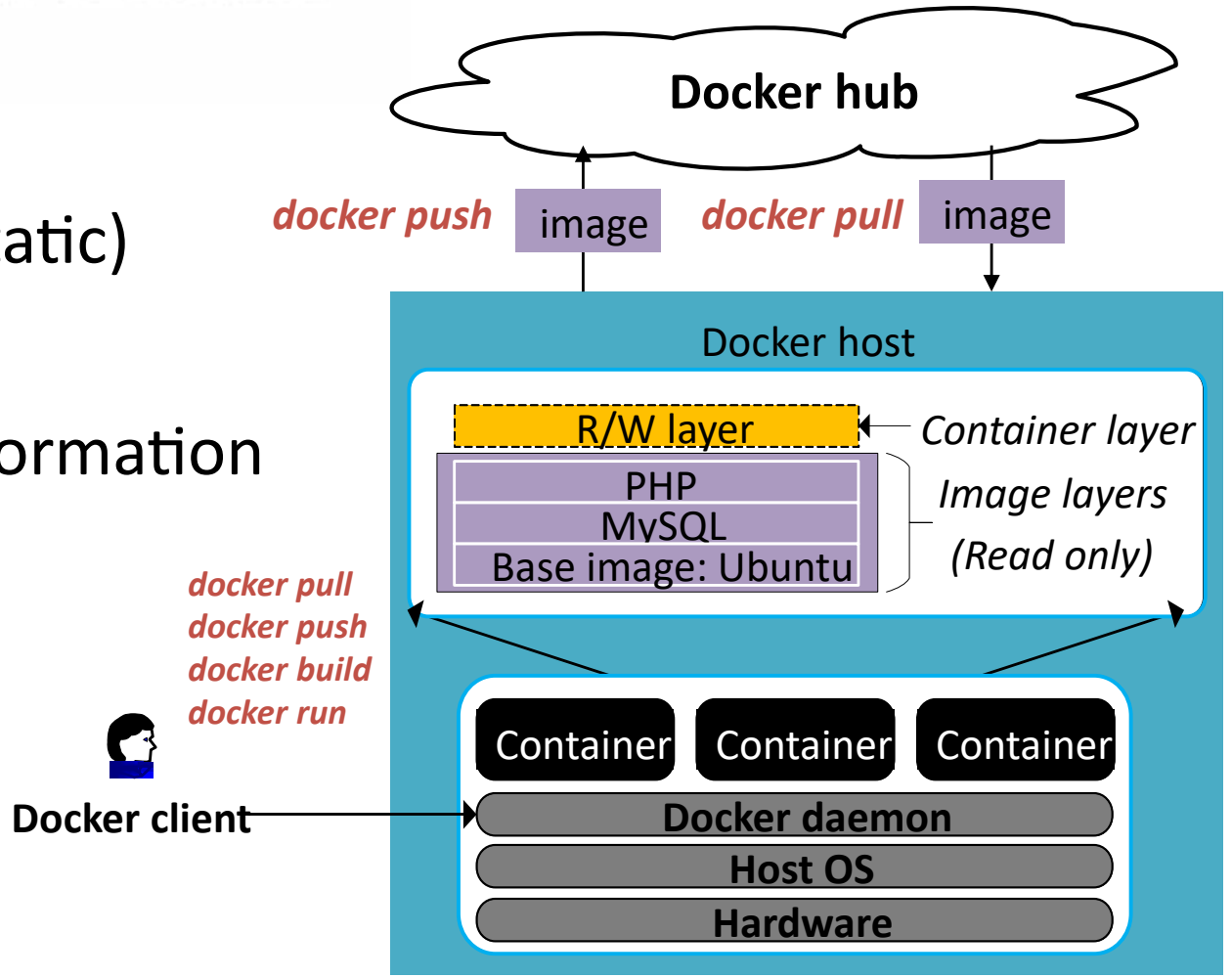
Image manifest(json)

```
"rootfs": {
 "type": "layers",
 "diff_ids": [
  "sha256:a5db9",
  "sha256:424de",
  "sha256:9bf110"
 ],
 .....
}
```

Ubuntu

MySQL

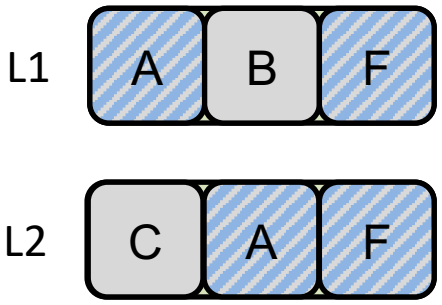
PHP



Deduplication in Docker

➢ Layer deduplication

- Coarse-grained: ***layer-level*** deduplication → Not Enough
- Layer compressed stored in registry
- Cannot deduplicate files across layers → increase redundancy
  - Executables, object code, libraries, and source codes
- New policy: remove inactive images for 6 months (***4.5PB***)



A & ***F*** can not be deduplicated
in two different layers with layer-level deduplication

Finer-grained deduplication is needed

PULL Latency

- Pulling packages accounts for **76%** of container start time[Harter, FAST'16]
- Uncompressed scheme will only worsen `GET` latency when pulling

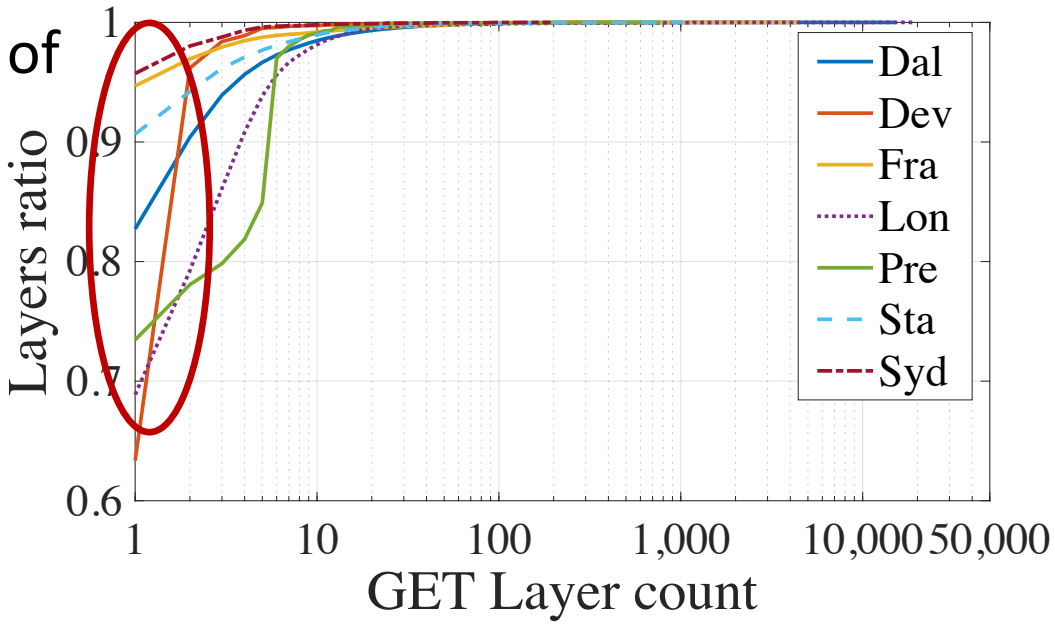
Technology	Dedup Ratio, compressed layers	Dedup Ratio, uncompressed layers	Latency increase if using uncompressed layers
Jdupes	1	2.1	36 ×
VDO	1	4	60 ×
Btrfs	1	2.3	51 ×
ZFS	1	2.3	50 ×
Ceph	1	3.1	98 ×

Need to consider latency

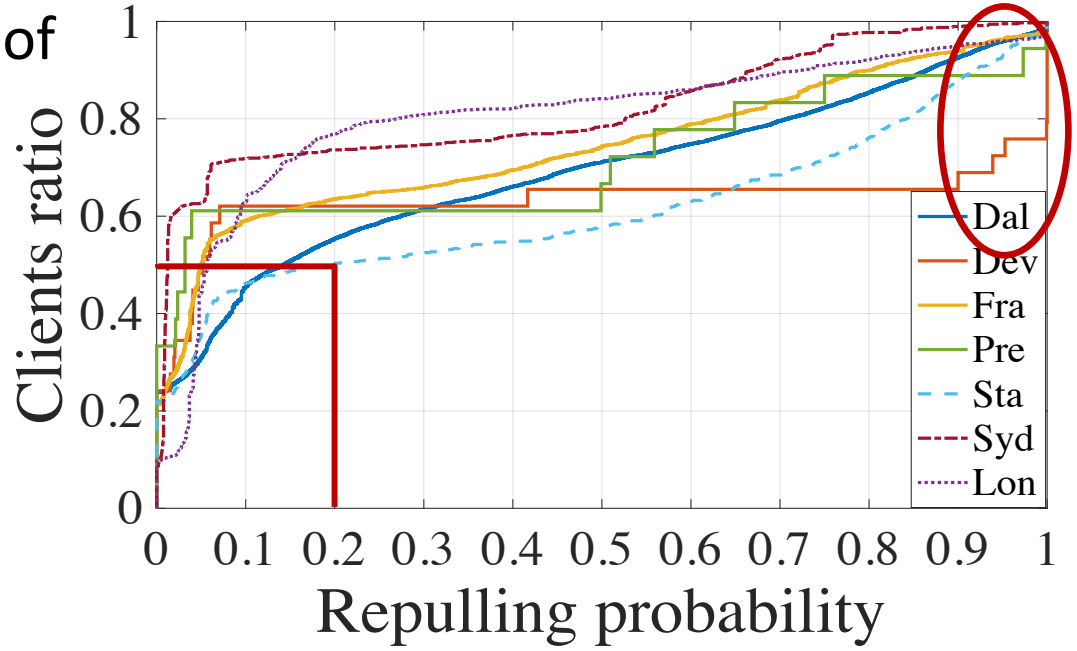
User Access Pattern Predictable

- The same client may pull most of layers only once
- Some layers are always re-pulled

CDF of



CDF of

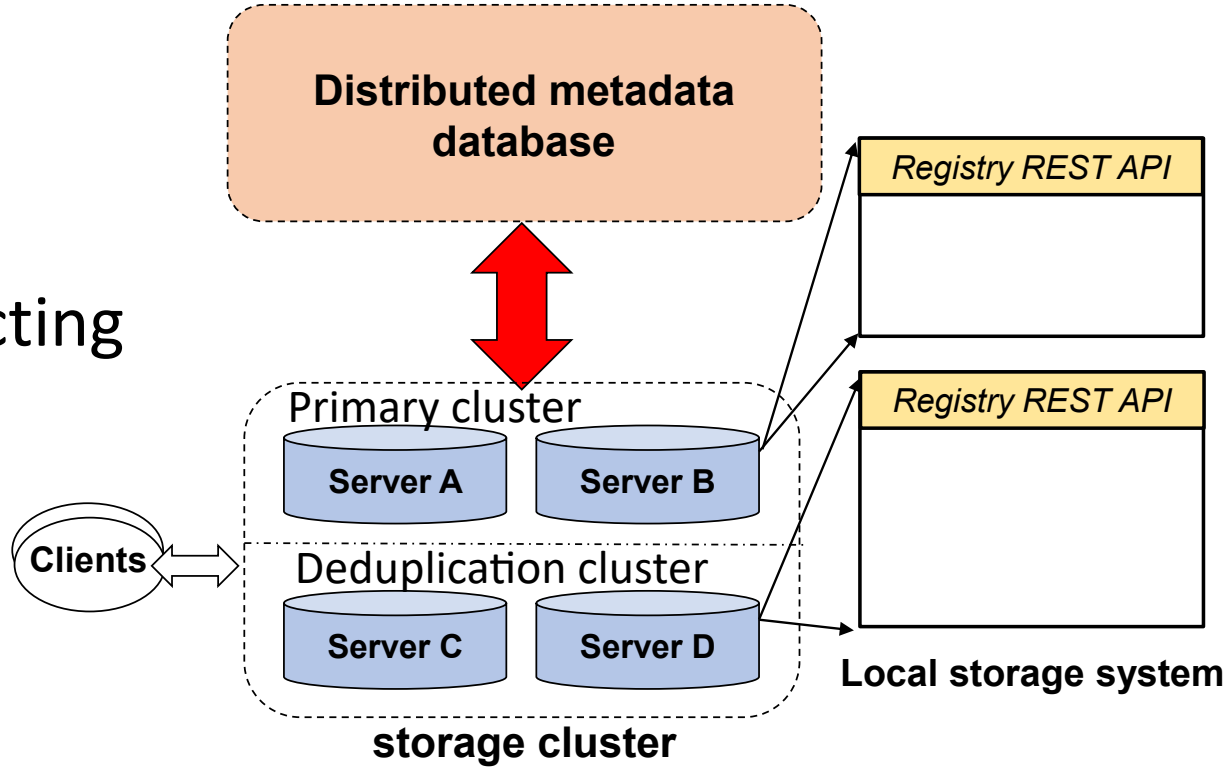


Motivation

- **Redundancy exists across in layers**
  - Finer-grained deduplication is possible
- **GET latency is already worse**
  - Deduplication can not prolong latency too much
- **User access pattern is predictable**
  - Cache can works under this predictable behaviors

Contributions

- DupHunter, a flexible and high performance deduplication system for registry
  - Replica deduplication mode
  - Parallel layer reconstruction
  - Layer prefetching/preconstructing based on user access pattern

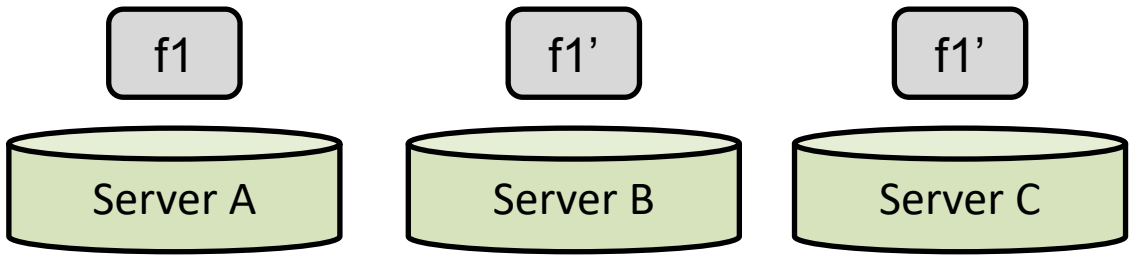


Replica Deduplication Mode

- **B mode n:** Basic deduplication mode n
  - Keep ***n*** replicas intact
  - Decompress and deduplicate files inside the ***R-n*** layer replicas (R is layer replication level, default: ***3***)

- **B-mode 3:** not deduplicate any layer replicas
- **B-mode 0:** deduplicate all layer replicas

- **S-mode:** Selective deduplication mode
  - Hot layers → more intact replica
  - Cold layers → deduplicated more aggressively



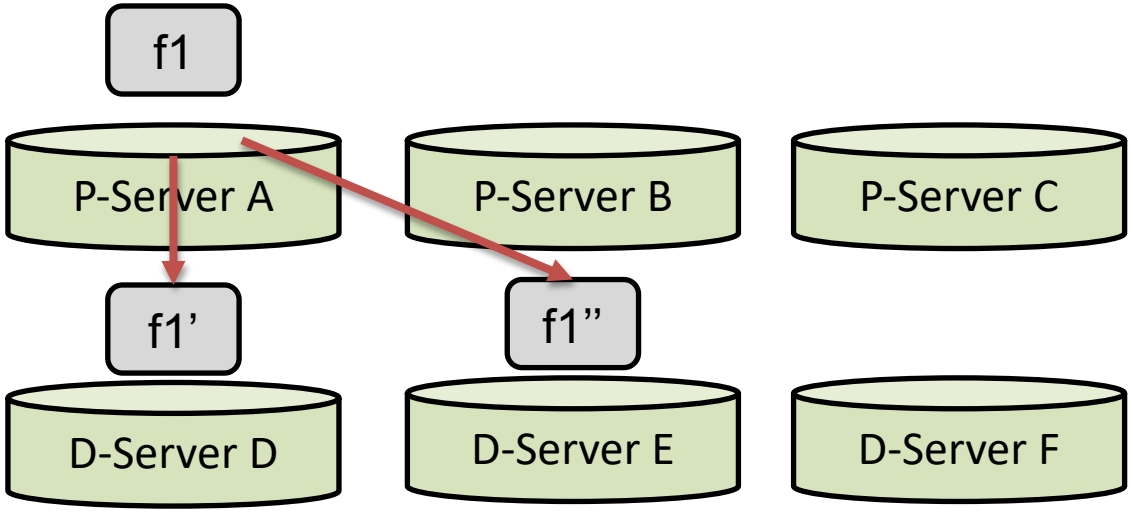
B-mode 1 when R = 3

- f1 → intact replica
- f1' → decompressed and deduplicated

Two-tier Cluster

- **P-server:**
  - Primary server for storing layer replicas and manifests
- **D-server:**
  - Deduplication server for deduplicating files

→ to isolate layers and deduplicated files (simplify)

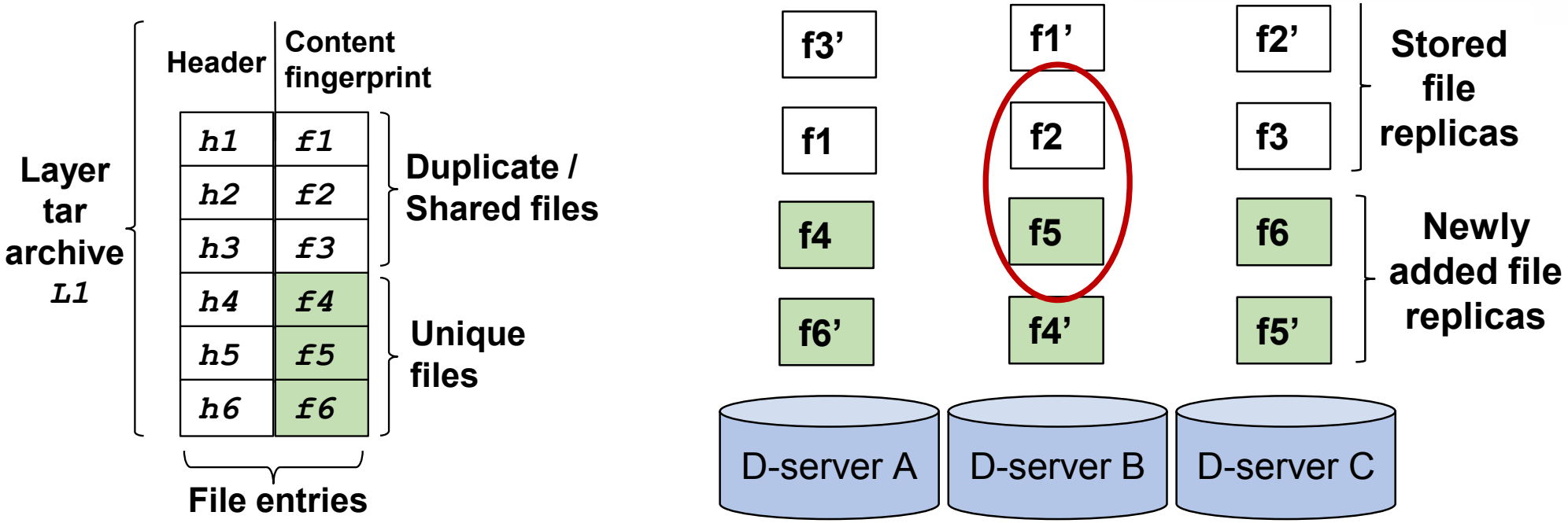


B-mode 1 when R = 3

- f1 → intact replica
- f1' and f1'' → decompressed and deduplicated

Two replicas on two different D-servers

Deduplication on D-servers



Distributed Metadata Database

File index

Id	r1	r2
f1	A:/../..	B:/../..
f2	B:/../..	C:/../..

Layer recipe
Id: L1
Master: A
Workers: [A, B, C]

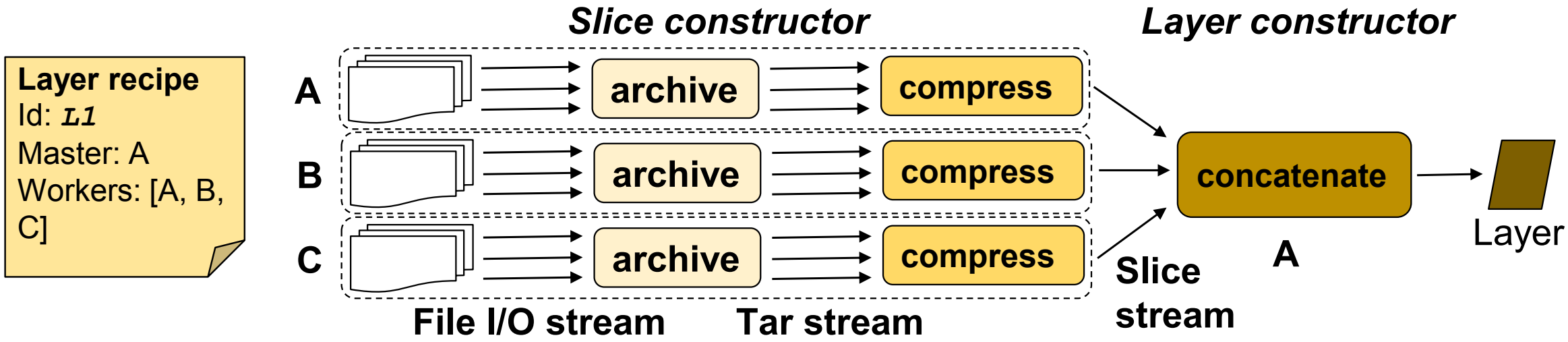
Slice recipe
Id: L1::B::P

Header	Content pointer
h2	f2
h5	f5

- **Slice:** set of all the files on a server belonging to a layer
- **B:** the primary slice on server B
- **P:** the backup level for primary slice

Parallel Layer Reconstruction

- According to layer recipe, master issues requests to workers to parallel layer reconstruction
- `cat file1.tgz file2.tgz > file1_2.tgz`

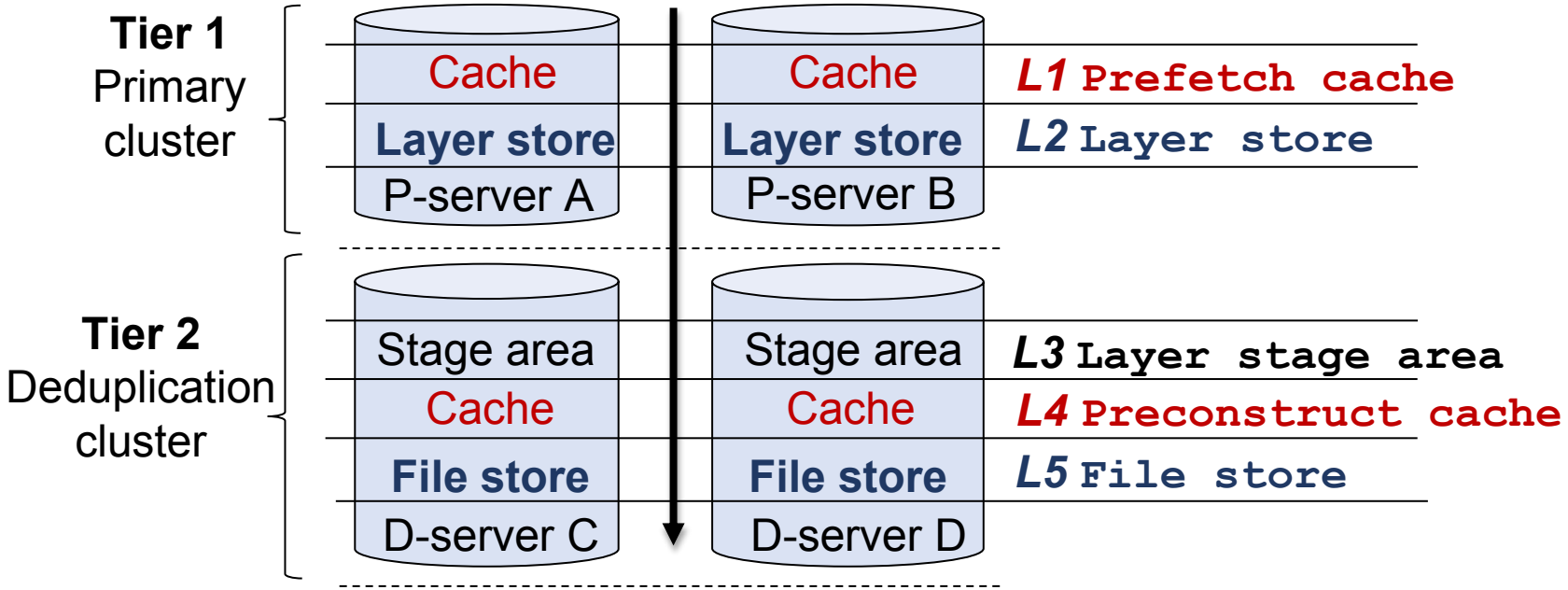


- If the layer is being assembling, its subsequent `GET` request awaits for its completion
- Multiple layer reconstruction is allowed

Prefetch and Preconstruction Cache

- **Cache for user access prediction**
  - **ILmap** (image ↔ sets of layers)
  - ***ULmap*** (user ↔ accessed layers and its count)

- **P-server:** In-memory layer prefetching cache
- **D-server:** On-disk layer preconstructing cache



Experimental Setup

- Workloads:
  - Traces from IBM registries: Dal, Fra, Lon, and Syd
  - Dataset downloaded from Docker Hub
- Schemes:
  - **Baseline**: no dedup with 3-way replication (***Bolt***)
  - **B-mode *n*:** *n (1-3)* replicas are preserved; *3 – n* deduplicated
  - **S-mode**: intact layer replicas proportional to the layer's popularity
  - **B-mode 0**: deduplicate all layer replicas
    - **GF-R**: global file-level deduplication with replication
    - **GF+LB-R**: global file-level deduplication and local block-level deduplication stored on *VDO*
    - **GB-EC**: global block-level deduplication under erasure coding **(6,2)**

- **B-mode 3:** not deduplicate any layer replicas
- **B-mode 0:** deduplicate all layer replicas

Dedup Ratio and Performance in Latency

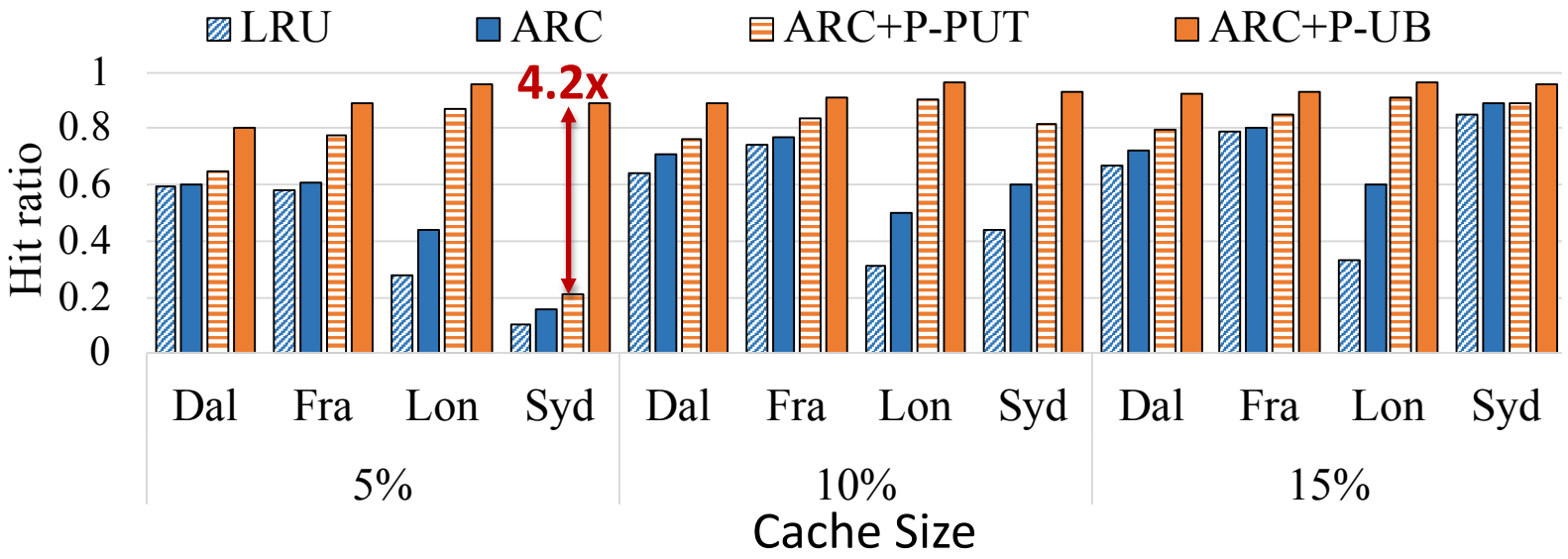
Mode	Dedup. ratio	Performance improvement (P-servers)
B-mode 1	1.5	1.6×
S-mode	1.3	2×
B-mode 2	1.2	2.6×
B-mode 3	1	2.8×
B-mode 0	Dedup ratio	Performance degradation (D-servers)
	GF-R (Global file-level [3 replicas])	
	2.1	-1.03 ×
	GF+LB-R (Global file- and local block-level [3 replicas])	
	3.0	-2.87×
	GB-EC (Global block-level [Erasure coding])	
	6.9	-6.37×

- **B-mode 3:** not deduplicate any layer replicas
- **B-mode 0:** deduplicate all layer replicas

- **B-mode 3** has **2.8x** improvement due to non-deduplication scheme.
- **6.9x** space saving under **GB-EC** mode.
- GF+LB-R is only slower ***2.87x*** while *VDO-only* scheme is slower 60x.

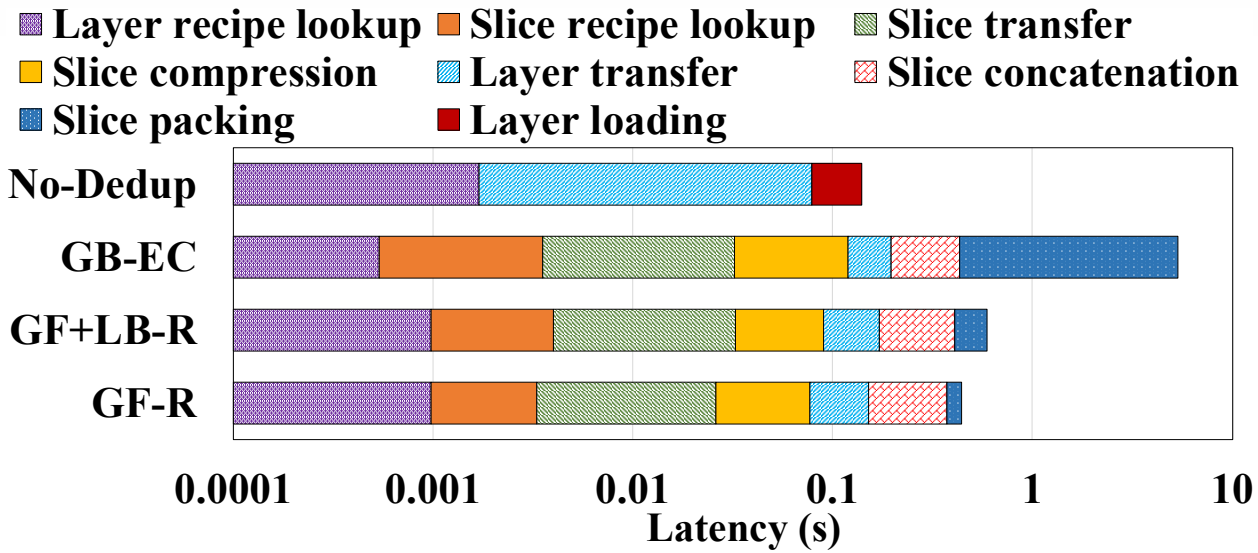
Prefetch Cache Hit Ratio

- **ARC+P-PUT:** prediction based on PUT requests
- **ARC+P-UB:** prediction based on user behaviors



- As cache size increases, hit ratios increases.
- **ARC+P-PUT** acts like a write cache, which is not practical due to its long layer reuse time.
- **ARC+P-UB** has **4.2x** improvements on Syd compared to **ARC+P-PUT** because **ARC+P-PUT** ignores that some clients always repull layers.

Layer Restoring Latency Breakdown

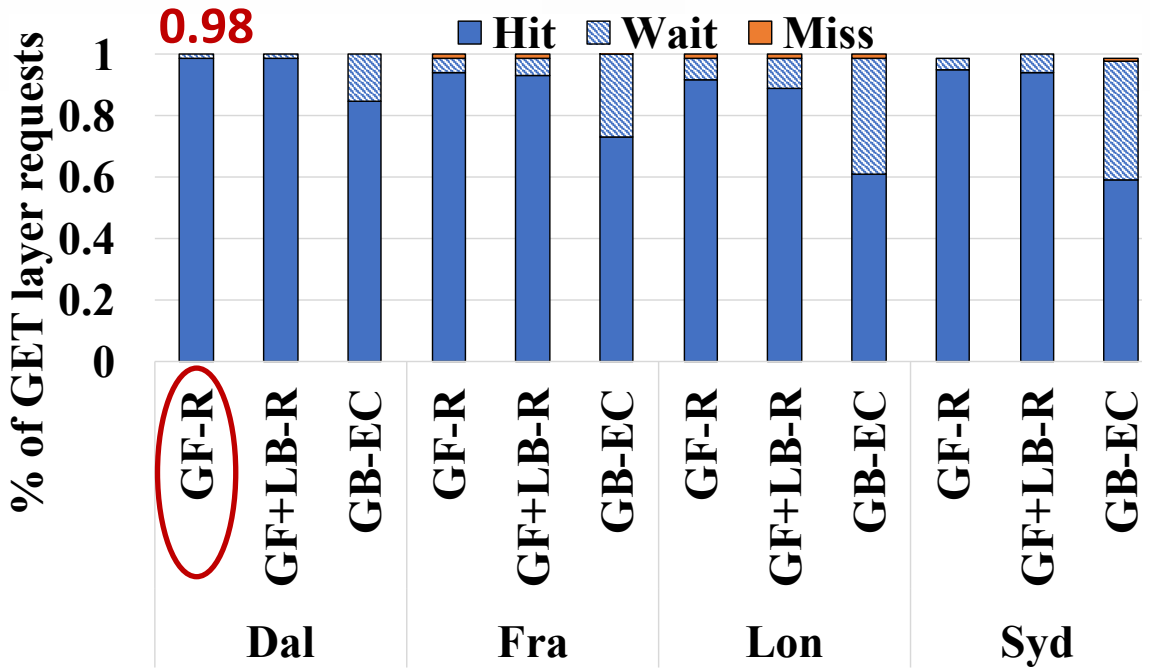


- **GF-R** increases by **3.1x** compared to **No-Dedup**. Half of layer latency → slice concatenation (cannot parallel).
- **GF+LB-R** requires additional reading process from local VDO device → additional overhead
- **GB-EC** has highest restoring overhead → File is split into four data chunks, distributed and deduplicated

Preconstruct Cache Hit Ratio

Cache size: **10%**

The higher for hit, the better



- **Hit:** layer requested is present in cache
- **Wait:** layer is being constructed
- **Miss:** Neither in cache nor being constructed

- **GF-R** has the highest hit ratio and lowest wait and miss ratios because it has lowest restoring latency.
- **39%** of GET layer requests are waiting for **GB-EC** → layers cannot be preconstructed on time.

Conclusion

- DupHunter provides a balance between storage saving and latency and exploits redundancy in images with user prediction
  - Replica deduplication mode
  - Parallel layer reconstruction
  - Proactive layer prefetching/preconstruction based on user's access pattern
- Space saving by up to **6.9x**
- `GET` Latency reduced by up to **2.8x**