

Content-addressed deduplication: a distributed-storage-system study

CS 4993 · Research specification

Harivansh Rathi

September 2026

Abstract

A content-addressed store identifies each chunk by a cryptographic hash of its contents, so identical data receives an identical identifier on every host. Deduplication in ZFS and dm-vdo is confined to one pool, so a fleet stores a shared chunk once per host and migrates a guest by copying its image. This study implements a content-addressed block backend for virtual machines beneath unmodified QEMU. We predict that a guest is provisioned or migrated by moving only its manifest, that two hosts store at most 55% of what two per-host ZFS pools hold, that a cold 4 KiB read from a peer's memory over TCP arrives before one from local NVMe, and that single-host capture is within 10% of ZFS fast dedup at equal block size. The testbed is two hosts with static membership, Linux guests, and single-digit terabytes, and the cost measured is the network round trip on the cold read path and, in fleet class, on the FLUSH path.

Contents

0	Thesis	3
0.1	Deduplication within a host	4
0.2	What is gained across hosts	4
0.3	What is paid	5
0.4	Hypotheses	5
0.5	Results	6
0.6	Scope	6
1	Architecture	7
1.1	Agreed baseline and scope	7
1.2	Implementation status · 14 September 2026	7
1.3	Components on one host	8
1.4	Watermark	8
1.5	Retained recovery and ownership	9
1.6	Write path	9

1.7	Admission, scheduling and resource limits	10
1.8	Compactor	11
1.9	Read path	12
1.10	Shared guest read memory [PROPOSED]	13
1.11	Store, index, and manifest	13
1.12	Protocol [AFTER C5]	13
1.13	Placement and the parameter k	14
1.14	Durability classes	14
1.15	Garbage collection (GC)	15
1.16	Provenance	15
2	Single host	15
2.1	Configurations	15
2.2	Chunk-size arms	16
2.3	Workloads	16
2.4	Metrics	17
2.5	Controls	17
2.6	Guest memory sharing [PROPOSED]	17
2.7	Payload placement [PROPOSED]	18
2.8	Census	18
3	Multiple hosts	18
3.1	Two placement modes	19
3.2	Provisioning	19
3.3	Migration	19
3.4	Synchronization after drift	20
3.5	Capacity	20
3.6	The local-class window	20
3.7	Measurements	20
3.8	The locality objection	21
4	Remote read	21
4.1	Where the time goes in a remote read	21
4.2	Transport probes	22
4.3	Method	22
4.4	Prefetch	22

4.5	Under a guest workload	23
4.6	The FLUSH round trip	23
4.7	RDMA on this testbed	23
5	Plan	23
5.1	Development checkpoints and research gates	23
5.2	Hardware	23
5.3	Schedule	24
5.4	Gates	24
5.5	Descoping order	25
5.6	Risks	25
5.7	Logistics	25
5.8	Future work	25
6	Prior art	26
6.1	Nearest systems	26
6.2	Remote fetch in prior systems	27
6.3	Objections already in print	28
6.4	What this study adds	28

0 Thesis

Addressing a virtual machine's data chunks by their content lets a fleet of hosts deduplicate, provision, and cache across host boundaries. At what scale does content-addressing your data become economically viable? This study measures those advantages and their cost on a multi-host cluster, and projects from them the fleet size at which a content-addressed store becomes advantageous. Existing solutions such as ZFS and dm-vdo also hash blocks for deduplication, but the hash serves only as a key into a table scoped to one pool while the block remains addressed by its location on disk, so no identity the table records is visible beyond a single host.

Our system has three pre-requisites A guest is provisioned or migrated by moving its manifest Each unique chunk is stored k times across the fleet instead of once per host A chunk many guests read is served from its owner's memory. Pages 03 and 04 measure each individually.

A cold read of a chunk another host holds costs one round trip on the network, the latency for this is measured in this study over NVMe-tcp as well as RDMA Durability before acknowledgment becomes a choice between this host's disk alone and a peer's disk as well.

The system is a content-addressed block backend under unmodified QEMU. Its scope, called the testbed from here on:

- Two hosts with static membership
- Linux guests
- Assumed single-digit terabytes scale

0.1 Deduplication within a host

Two guests that install the same package hold equal bytes that no clone or snapshot can share, because neither copy descends from the other.

A deduplication table shares them. OpenZFS keeps one per pool, the DDT, and Linux has had dm-vdo in the mainline kernel since 6.9. Both hash every block and share equal blocks at one fixed, aligned size: 4 KiB for dm-vdo, and the volblocksize of a ZFS zvol, 16 KiB by default. This study calls the guest's 4 KiB unit a block and the store's unit a chunk, whether the chunk's boundaries are fixed or content-defined.

Nearly everything a Linux guest writes is 4 KiB aligned. ext4 uses 4 KiB blocks, partitions start at 1 MiB, and package managers write whole files. Jin and Miller found on VM disk images that fixed-size chunks reach nearly the same deduplication ratio as content-defined chunking (CDC), which places chunk boundaries by the bytes themselves.

We therefore predict that on one host the backend stores within 10% of what ZFS fast dedup stores when its chunk size equals the zvol's block size. Page 02 tests this as hypothesis 1.

None of this reaches across hosts. The DDT is per pool, dm-vdo has no replication, and zfs send has not carried a deduplicated stream since OpenZFS 2.0. A fleet of N hosts, each with its own table, stores a chunk shared by all of them N times. Moving a guest to another host sends every block of its image, whether or not the destination holds an equal one.

Shared-storage systems deduplicate across hosts by placing every write on the network before it is acknowledged. Ceph RBD with TiDedup is the open example. Page 06 lists the systems on either side.

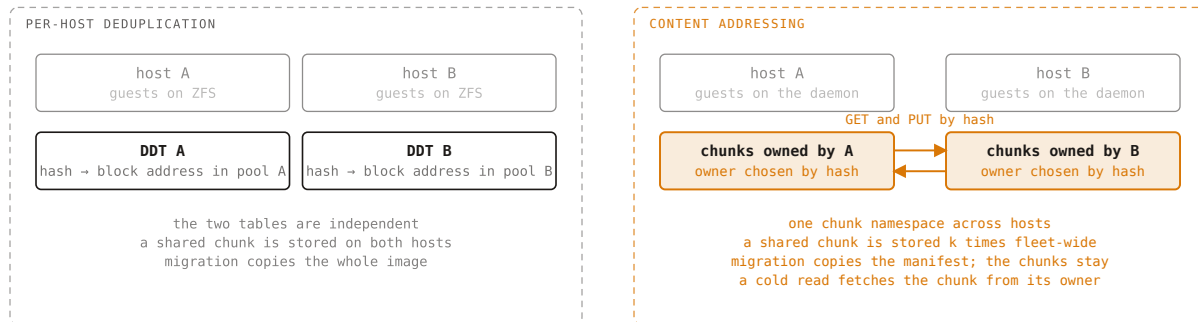


Figure 1: Left: a deduplication table indexes blocks within one pool. Right: a hash is the same name on every host, so placement, transfer, and the cache key follow the content.

0.2 What is gained across hosts

Each consequence below is measured on pages 03 and 04.

Transfer. Provisioning a guest from an image whose chunks exist in the fleet moves the manifest and no chunk data. Migrating a guest moves the manifest plus the bytes written since the last compaction.

Capacity. The fleet stores each unique chunk k times rather than once per host, and each host's index holds entries only for the chunks it owns, plus surplus copies until their owner acknowledges them.

Cache. Every host sends its reads of a chunk to the same k owners. We predict that a chunk many guests read is therefore hot at its owner, and page 04 reports the owner's hit rate. **On the published figures page 04 stacks, a chunk in a peer's memory arrives before a chunk on the local disk.** Hypothesis 3 tests this.

0.3 What is paid

Three costs come with any post-process deduplicating store, and each is measured in this study:

- write amplification, because every surviving byte is written to the staging log and again to the store
- compactor interference, because compaction shares the guest's disk
- index memory, one entry per chunk in RAM

One cost belongs to distribution alone. For a chunk this host does not hold, the network is on the read path. Page 04 measures that read over TCP and over RDMA, from the peer's memory and from the peer's NVMe. It then measures how much prefetch, described on page 01, removes.

One cost is a tradeoff the design makes on purpose: durability before acknowledgment. In local class, the default, a FLUSH is acknowledged after `fdatasync` on this host, the contract a local disk gives, and a host lost before compaction has shipped its bytes loses them. In fleet class the FLUSH also waits for a peer's `fdatasync`, as the hyperconverged products on page 06 do, so `fsynced` bytes survive the loss of this host. Page 01 defines the classes and page 04 measures the round trip fleet class adds.

0.4 Hypotheses

Each hypothesis states a metric with its conditions, a comparator, a threshold, the source of the threshold, and what a miss would show. Thresholds are frozen at the end of week 2, after R0 has measured the testbed's `fdatasync` and media times, and do not move after that.

1. Single-host parity. Bytes stored by the backend after compaction and sweep, under the fleet replay at fixed 4 KiB and 16 KiB, are within 10% of the bytes ZFS fast dedup stores at the same `volblocksize`. Bytes stored in each chunk-size arm are within 10% of the census prediction for that arm. Guest write and read p99 at 4 KiB QD1, with the compactor idle and again with it active, are within 20% of a raw file on XFS. The 10% is the alignment argument above plus record headers, since the sweep runs before every capacity number. The 20% is the passthrough bound of gate G1 plus an equal allowance for the log append and compactor interference. A miss on capture would show that fixed aligned chunks lose duplicates a Linux guest produces. A miss on p99 would show that the host daemon (not deduplication) is the cost paid.

2. Transfer and capacity across hosts. Bytes on the wire to provision a guest are within 10% of its manifest size, and to migrate one within 10% of the manifest plus the staging tail, against the allocated image size that `zfs send` or `rsync` moves. Bytes sent to synchronize two drifted guests are within 10% of the census's unique-byte count for the pair. Bytes stored on both hosts in partitioned mode, after the sweep, are at most 55% of the bytes two per-host ZFS pools hold for the same guests. The 10% on transfer covers framing and HAS replies, since no chunk an owner holds is sent by design. The 55% is one copy of the unique set instead of two, because guests cloned from one image give the two pools nearly the same unique set, plus five points for record headers and manifests. A miss on transfer would mean chunks were sent that an owner already held, a HAS or fence defect. A miss on capacity would mean the two pools shared less than the census predicted, which the census would show first.

3. Reads over the wire. For a 4 KiB read at QD1 whose chunk is not in the local cache, over the daemon on kernel TCP: Served from the owner's memory, guest-visible latency is lower than the same read served by the daemon from its own NVMe. Served from the owner's NVMe, it is at most 40% over the local read. With reads in flight at or above the bandwidth-delay point, remote sequential throughput is within 10% of local. In a partitioned boot storm of 16 guests with profile prefetch, guest p99 is within 25% of the same storm in replicated mode. The kernel `nvme-rdma` probe, which stands in for a daemon over RDMA, serves the owner's NVMe at most 15% over the local read. The probe is not the architecture, and its number is reported as the floor the `ibverbs` arm could approach. The thresholds are the literature stack on page 04: about 80 μ s of media, plus 20 to 30 μ s for a userspace daemon over kernel TCP and about 12 μ s for kernel `nvme-rdma`. A miss on the first part would show that the kernel stack or the daemon's wakeup costs more

than the media. A miss on throughput would show that the fabric bounds it. A miss on the boot storm would show that prefetch does not hide the remote read under a real access pattern.

4. Durability before acknowledgment. Write p99 at 4 KiB QD1 in fleet class is within 3x of local class over TCP, and within 2x over RDMA if the ibverbs arm lands. The window of local class, the seconds between a FLUSH acknowledgment and the durability of those bytes at their owners, is reported as a distribution under the fleet replay. The 3x is one round trip plus one peer fdatsync alongside the local fdatsync, on the page 04 figures and an fdatsync near 40 μ s (NEED DATA; measured on the testbed drive in week 1). A miss would show that the journal path rather than the transport is the cost. The peer's fdatsync time is reported separately so the two can be told apart.

0.5 Results

The system. A content-addressed block backend for VMs under unmodified QEMU on a stock Linux kernel, over kernel TCP, with source, configuration, and the scripts that produce every table.

The single-host table. The backend against ZFS fast dedup and a raw file on XFS: bytes stored, guest p99, write amplification, and index memory, at three chunk sizes. Hypothesis 1 is decided here. The capture against index memory curve across the three arms is reported without a threshold, because no prior curve on NVMe exists to bound it.

The multi-host table. Bytes moved to provision and to migrate a guest, bytes sent to synchronize two drifted guests, fleet bytes stored with one copy per chunk, and index bytes per host, each against what `zfs send` or `rsync` moves and what two per-host ZFS pools hold.

The remote-read measurement. A content-addressed chunk fetched from a peer under a VM block device, at microsecond resolution, over the daemon on kernel TCP and over NVMe-oF on TCP and RDMA, from the peer's memory and from its NVMe, with and without prefetch.

The durability trade. Local class against fleet class on the same hardware: the write latency fleet class costs per transport, and the seconds of acknowledged data local class puts at risk.

0.6 Scope

The study covers hosts that serve guests from local flash, from a small homelab setup up to rack scale (storage arrays and hyperscale economics are out of scope). Hosts hold each other's chunks, which couples the failure domains of compute and storage that shared-storage designs keep apart. This study measures what that costs on the read path and does not model its availability.

The testbed is two hosts with static membership. Membership changes, failure detection, rebalancing, authentication and encryption on the wire, measurement on more than two hosts, and concurrent garbage collection are out of scope, and none affects a number reported here.

Each image has only one writer. Ownership state, the root record that names the writer and carries a generation number, is held on both hosts. Two hosts form no quorum, so failover of a lost writer is a scripted decision and not automatic. The study migrates disks only. Memory migration is QEMU's own live migration.

The guest contract is `virtio-blk` with a volatile write cache. An acknowledged FLUSH is durable; ordinary write completion does not promise power-loss durability.

Equal BLAKE3 hashes are taken to mean equal bytes. A sample of matches is verified byte for byte and the sample size is reported. The store is trusted infrastructure, so deduplication side channels are documented and excluded.

Experiments run at single-digit TB, and larger figures are projections from measured constants, labeled as

such.

1 Architecture

In local class the network is on the read path only, and only for a chunk this host does not hold. In fleet class it is also on the FLUSH (fsync) path, once per FLUSH, to only one fixed peer.

1.1 Agreed baseline and scope

The C0–C5 implementation is one host daemon serving multiple private images: local durability, fixed 4 KiB chunks, a separate copying store, COW manifests, a bounded clean read cache and quiescent GC. Stock QEMU and normal guest caching remain in place. On this single host, $O = D$.

The diagrams also show the later distributed design: peer GET/PUT, placement, surplus repair, migration and fleet-class journaling. Those paths are outside C0–C5. The 16 KiB/CDC compactor arms, prefetch, DAX/shared guest memory, guest-buffer zero-copy and in-place payload placement require their own experiments; the baseline starts with prefetch off.

This page summarizes the architecture agreed in PR #26. The C0 design defines the full local contract and failure traces; storage-format.md fixes the exact bytes. Linked implementation notes refine that contract. The 16 MiB manifest-page cache below is one such later implementation choice.

1.2 Implementation status · 14 September 2026

Checkpoint	Validated	Remaining
C0–C3	Agreed design; build-bound validator; one-copy packed append; four queues, FLUSH ordering and retained live recovery.	Accepted development checkpoints.
C4	Store, manifests, compaction, snapshots/clones and GC. All 40 functional scenarios passed and independently verified, including six compaction crash cuts.	Final allocation audit before aggregate closure.
C5	Two private ext4/SQLite guests, retained restart and fresh boots; shared caches, scheduling, all 11 pressure stages and the consolidated 41-scenario inventory passed independent verification.	Final host/guest memory accounting and dedicated-media repetitions.

The initial implementation merged in PR #27. Subsequent atomic increments validate scheduling, telemetry and competing workloads, including staging backpressure and a burst above measured drain. The latest-main integration passed all 41 C5 scenarios and independent verification on b8399ad, plus Rust and Nix/guest CI. The final Markdown receipt retains earlier failures and their corrections. Final memory accounting remains open. Since that run, PRs #48, #49 and #53 changed capacity waiting, compaction preparation and read admission; the C5 suite has not been rerun on those changes. TODO.md tracks completion; the implementation status lists the remaining work and evidence.

Spark/KVM checks establish development correctness under recorded failure models. Dedicated-media repetitions, physical power-loss experiments and research gates G1–G6 remain pending. This dated snap-

shot does not establish CI, merge or live deployment.

1.3 Components on one host

The guest runs a normal Linux filesystem on a virtio-blk device under stock QEMU. Guest caching remains enabled. Applications can choose guest `O_DIRECT` independently of the host IO mode.

QEMU configures the device and passes its queues and guest-memory mappings to one daemon per host over `vhost-user-blk`. Mapping guest memory adds no payload copy. The daemon submits storage IO through Linux `io_uring`. The device advertises a 4 KiB logical block and validates whole-block requests. Virtio addresses remain in 512-byte sectors. Host buffer alignment is checked separately.

QEMU reconnects after a daemon restart. Recovery preserves completed writes and replays requests still in flight without overwriting newer data. Concurrent retained recovery has passed the development checkpoint; the implementation plan distinguishes it from the preserved serial reference.

1.4 Watermark

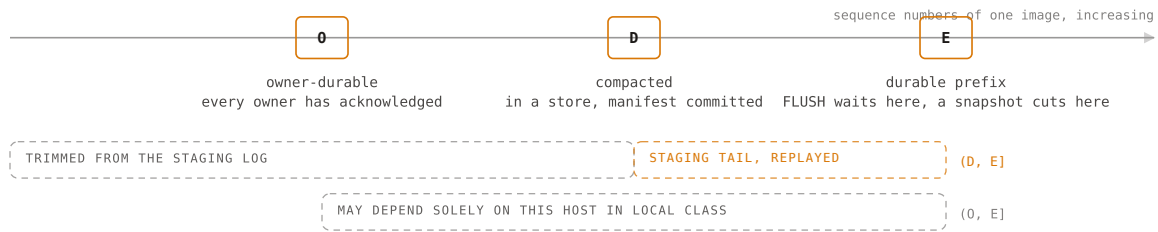


Figure 2: O tracks owner durability, D compaction and E the durable prefix. Data in $(O, E]$ may depend solely on this host in local class.

Three durability watermarks describe each image. E is the highest sequence number with no unconfirmed append before it. In local class confirmed means on local NVMe. In fleet class it means on the journal peer too. D is the highest sequence number whose chunks are durable in a store, at their owners or as surplus copies on this host, and whose manifest entries are committed. $O \leq D$ is the highest sequence number whose chunks are durable at every owner. O equals D except while a surplus copy stands in for an unreachable owner. FLUSH waits until E covers its captured boundary. A snapshot cuts at E. The staging log is trimmed below D; discarded regions can be reclaimed by the drive. Durable recovery replays $(D, E]$. A daemon-only restart also preserves completed writes beyond E while the host and device remain live. Owner confirmation alone does not protect a chunk whose only owner is the lost host. E never skips a hole, because a maximum over confirmations forgets the append still in flight, and that is the answer that loses acknowledged data.

The committed manifest and staging must agree after a crash. A valid durable root supplies data through D; staging supplies newer mappings. Reclaimed payload below D is not required for replay. A selected complete root with missing or corrupt durable chunks is an error. Re-running compaction over the replayed extents yields a manifest whose every offset maps to the same bytes. It need not yield the same chunk boundaries under CDC, and the sweep reclaims the orphans of the first run. `kill -9` at any point, then this replay, must pass `fio --verify` before any number from the daemon is reported. The log's torn tail is tested in both shapes, a shortened file and a partial record followed by preallocated zeros. Three more cases have tests because each is a defect the author met in a prior implementation:

- an empty discard that acknowledged a sequence number nothing wrote and wedged the next FLUSH

- a FLUSH that must cover writes completed on any queue, checked with a multi-queue test and a negative control that shows the test can see the reordering
- a daemon that stops answering, which leaves the guest in D-state because virtio-blk installs no timeout handler

1.5 Retained recovery and ownership

The local runtime also records P , the ordered publication frontier, in QEMU's retained `INFLIGHT_SHMFD` trailer before guest completion. A completed WRITE can be beyond the last durable FLUSH. On daemon replacement, the same live guest and retained FD preserve request identities, `PREPARED/ACTIVE` transitions and the prefix through P . Cold recovery instead relies on synchronized durable state; retaining P is not a power-loss guarantee.

Replay reuses original identities and mutation order. File locks prevent repair while old kernel IO owns a segment. Reads pin their captured staging/manifest view; reset, cancellation and timeout retain buffers and mappings until kernel ownership ends. A terminal storage failure publishes `FAILED` before `IOERR` and prevents later success from crossing that failure. The C0 design specifies each transition and interruption case.

1.6 Write path

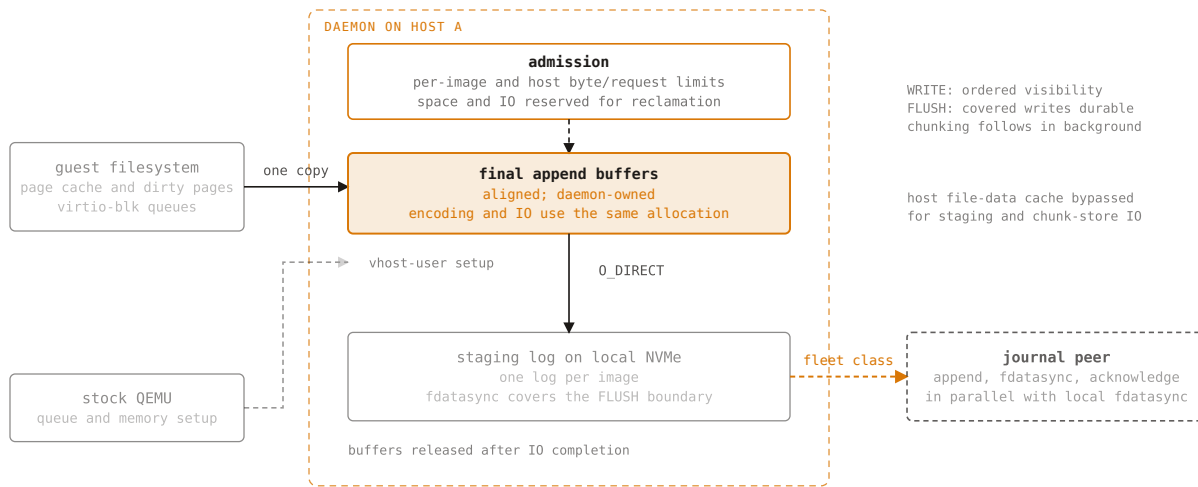


Figure 3: Target write path: one host payload copy into a bounded append buffer. WRITE completes after staging IO and ordered visibility; FLUSH waits for durability.

Each WRITE copies guest payload once into its final append buffer. Record encoding and storage submission use the same allocation. The buffer remains owned until its IO completes. The daemon bounds the append-buffer pool by bytes and request count, with admission limits per image. The copy count describes this host write path; guest application copies, peer transfers and later storage writes are counted separately.

An aligned guest buffer could instead feed vectored direct IO without a daemon payload copy. That path would need stable guest bytes and valid mappings until IO completion, a compatible record layout, and recovery across resets. The baseline uses one owned buffer for a stable, aligned record. Linux `io_uring`(7)

Guest writes append at block granularity to a staging log on local NVMe, one log per image. Sequence numbers and log positions are assigned together. Mutation publication follows sequence order even when stor-

age IO completes out of order; later writes wait for earlier mutations before index visibility and guest completion. The index holds block-to-log offsets and ordering metadata, with no retained payload. A WRITE completes after its staging IO and ordered index update. A later read observes that write or a newer write to the same block.

The device requires FLUSH support. FLUSH captures a per-image sequence boundary covering writes completed on every queue, waits for the covered appends, and calls `fdatasync` before acknowledgment. Local class waits only for the local log. Fleet class also waits for the journal peer, as defined below. Hashing and chunking run after log durability, outside the FLUSH path.

Staging and chunk-store payload IO use `O_DIRECT` to bypass the host file-data page cache. Direct IO still requires synchronization for persistence. Buffer addresses, file offsets and lengths satisfy the backing filesystem's direct-IO alignment requirements. Linux `open(2)` A waiting FLUSH starts `fdatasync` as soon as its covered appends finish, without a batching delay. Later writes wait to submit until the active sync finishes; a FLUSH with a higher boundary follows those writes in the next cohort. An idle sync every 50 ms advances durability for writes without a guest FLUSH.

The governor limits allocated staging space per image and across the host. It reserves output space and minimum IO service for compaction and manifest commits before admitting more writes. When append exceeds safe reclamation, admission slows before consuming that reserve. Progress deadlines and the recovery or failure action are recorded before each run. Storage errors return `IOERR`; lack of space does not authorize admission beyond the reserve. The experiment records the point where pressure engages, per-guest latency and the time to drain after writes stop.

1.7 Admission, scheduling and resource limits

The agreed scheduler uses per-image FIFO admission and byte deficit round robin across images with a 1 MiB quantum. At the host, one of every four ready bulk submission opportunities is reserved for compaction; three favor demand IO, and unused opportunities can be borrowed. FLUSH/control bypass bulk work and have their own reserve. This is a service policy; guest fairness and latency still require measurement.

Bound	Development default
Requests / queues	1 MiB payload; 4 queues $\times$ 256 entries; 128 outstanding requests per image, 1,024 per host.
Read requests	Separate pool: 8 per image, 64 per host.
Descriptor snapshot quota	6,297,600 bytes per frontend, within foreground metadata; actual spans allocate on discovery.
Control reserve	8 requests / 64 KiB per image; 32 requests / 256 KiB per host.
Append / read buffers	Each pool: 8 MiB per image, 64 MiB per host, including retained allocations.
Clean chunk cache	256 MiB per host; read-fill LRU, no automatic write admission.
Metadata	256 MiB per host: 128 MiB foreground, 128 MiB reserved for compaction. The 16 MiB manifest-page limit is inside foreground metadata.
Segments / staging	64 MiB segments; allocated staging capped at 256 MiB per image, 1 GiB per host.
Compaction	One worker; at most 1 MiB input payload and 128 MiB new manifest pages per transaction; 100 ms settle, 1 s maximum durable-version age.
Deadlines	Capacity waits do not expire; 30 s submitted IO; 60 s recovery.

Reserve $R = 3S + M + 16$ MiB for background progress, where S is segment size and M the manifest

transaction cap: 336 MiB at these defaults. Account allocated bytes plus accepted promises. Start pressure compaction at 75% of a staging cap; stop foreground admission at the cap or before invading the disk reserve; resume below 60% when all budgets permit. Unique live data can exhaust usable capacity and return ENOSPC/IOERR. These are development settings, not measured optima; record every override.

1.8 Compactor

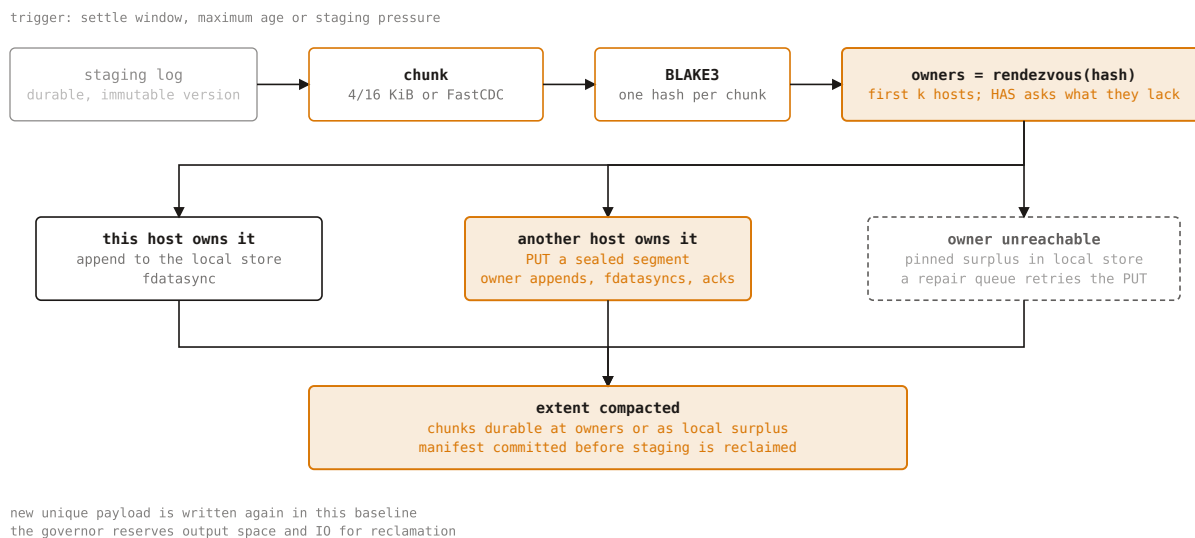


Figure 4: Background compaction in the copying-store baseline. New unique payload is written to a chunk store before staging is reclaimed; this is separate from the ingress copy count.

A background pass reads settled extents from the staging log, cuts them into chunks, hashes each with BLAKE3, and skips any hash that every current owner already holds and has fenced. A copy in a cache does not count as held. Chunking is fixed 4 KiB, fixed 16 KiB, or FastCDC with boundaries snapped to 4 KiB (one per measurement arm on page 02). Normally an extent is compacted after a settle window without writes. A maximum age or staging pressure forces a pass over an immutable version even if the guest keeps overwriting that extent. The window is a parameter, and its effect on chunk traffic is measured. A discarded or zero-filled range names no chunk and consumes no store payload. Range updates still require index and manifest work proportional to the affected mappings. A read of such a range returns zeros.

Rendezvous order of the hash names the k owners (Placement, below). If this host is an owner, the chunk is appended to the local store and made durable with fdatsync. Otherwise it goes to each owner in a sealed segment of many chunks, which the owner appends, fdatsyncs once, and acknowledges. **An extent counts as compacted at D, when its chunks are durable in a store, at their owners or as surplus copies here, and its manifest entry is committed. It counts as owner-durable at O, after every owner's acknowledgment.** If an owner is unreachable, the compactor appends the chunk to the local store as a surplus copy, pinned until that owner acknowledges it later. A repair queue retries the send. In local class, durable surplus copies and the manifest commit allow log reclamation during an owner outage while local capacity remains. Surplus copies consume the same disk budget. The sweep reclaims them after owner acknowledgment. A chunk the compactor has produced stays pinned, in the staging log or in a store, until the manifest commit that references it is durable. An owner never reclaims a chunk it acknowledged before that fence. The staging log is therefore the write-ahead log for every chunk this host produces, wherever the chunk might end up.

The compactor releases append and FLUSH locks before chunk IO, manifest IO or owner RPC. A test slows the store to one second per append while the log remains healthy and checks FLUSH against its recorded

deadline.

CDC over a dirty extent re-chunks from the last settled boundary before it to the first boundary after it that agrees with the existing cut. Two published properties make the rule exact (LBFS locality; Xet's boundary reset), and it is why CDC never runs on the hot path. One aligned write can move every boundary in its neighborhood.

1.9 Read path

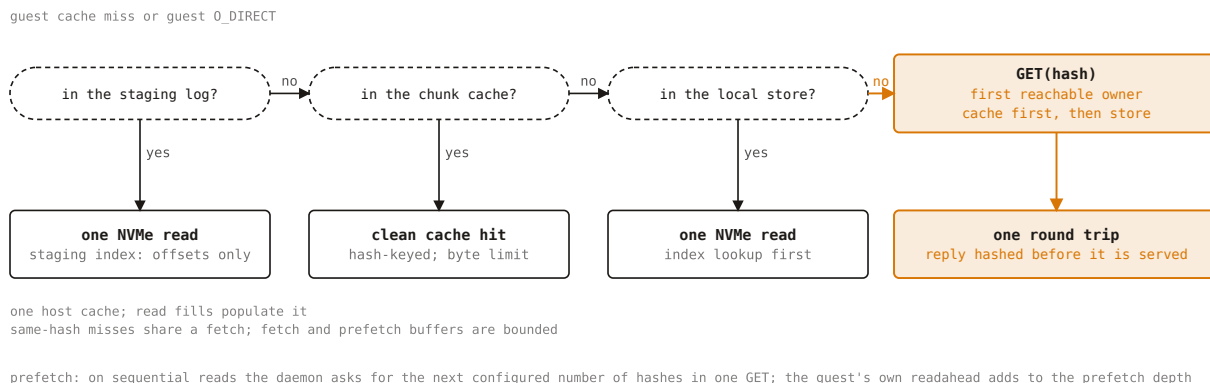


Figure 5: Reads after a guest-cache miss or guest direct IO. The host cache is clean and bounded; outstanding fetches have separate byte and request limits.

A guest-cache miss or guest direct read reaches the daemon. The staging index resolves fresh data first. For settled data, the manifest supplies a hash for the chunk cache, then the local store if this host holds the chunk. Otherwise the daemon sends GET to the first reachable owner in rendezvous order. The owner answers from its cache if the chunk is hot and from its store otherwise. Every chunk that arrives over the network is hashed before it is used, so a wrong or corrupt reply is detected and never served. A record read from a store is checked against its inline checksum. Fresh data is served without indirection. Settled data incurs the manifest lookup, the index lookup, and, if the chunk is remote, one round trip. GET uses separate connections and has priority over bulk PUT. Disk scheduling preserves the compactor service reserved by the governor.

One daemon-owned LRU chunk cache serves the host, keyed by hash and bounded by bytes. Read fills populate it; writes do not automatically enter it. Eviction drops a cached copy without changing durable storage. Evicted bytes remain charged until the final reader releases them. Coalesced reads retain the original fetch credit through the final shared reader; leaders and waiters are bounded. A fetched chunk this host does not own lives in that memory cache only. Concurrent misses for one hash share a fetch. Fetches and prefetch have bounded buffers and request counts, with demand reads served first. A disk tier for fetched chunks, as Liquid had, is a knob measured only if time remains, since page 04 predicts a refetch from a peer's memory costs less than a local disk hit.

Prefetch is the daemon issuing the next configured number of hashes from the manifest in one GET when it sees sequential reads, and optionally replaying a recorded boot profile. The guest's own readahead is left at its default and adds to that depth. Page 04 sweeps the prefetch depth; its parameter is separate from the publication frontier P above.

1.10 Shared guest read memory [PROPOSED]

An optional virtio-pmem frontend would expose one shared, immutable filesystem image through direct access (DAX). File-backed virtio-pmem uses the host page cache and bypasses the guest file-data cache. An OverlayFS upper would retain normal guest caching and the same private virtio-blk write contract. Copied-up files would read from that upper. The mounted lower must remain unchanged.

The first experiment uses a prepared local image. Mapping CAS chunks into guest address ranges, publishing new content, and fetching remote misses remain separate work. Sharing an existing image establishes frontend sharing; sharing independently produced equal content would establish the CAS-specific benefit. Page 02 defines the comparison.

1.11 Store, index, and manifest

The baseline local store is an append-only log of records (length, hash, checksum, bytes), separate from staging, and is authoritative for the chunks this host owns. Page 02 compares this layout with retaining and indexing surviving staging payloads in place; the guest contract is the same in both. The index maps hash to store offset, lives in memory, and is rebuilt by scanning the store without re-hashing, because the hash is inline. Its bytes per TB is the constant the chunk-size arms measure. In partitioned mode a host indexes the chunks it owns plus any surplus copies awaiting an owner, so per-host index memory is k/N of the fleet's once the repair queue is empty. An index entry is added only after the data it points to is durable, at every fence. The manifest, one per image, is a copy-on-write tree from disk offset to chunk hash, packed in offset order. A root commit becomes durable after its referenced chunks and before staging reclamation. It lives with the guest's host and moves when the guest does.

The local manifest uses checksummed 4 KiB COW B+tree pages, height at most eight, and fixed COMMIT pages identifying generation, root and D. Chunk data is synced before the manifest commit, and D publishes before covered staging can be reclaimed behind all read/replay pins. A shared manifest-page cache keys verified pages by incarnation, COMMIT end and offset. Cached bytes retain no root/file pins; every hit passes the checked lookup descent.

A local snapshot pauses mutation admission, drains and FLUSHes, and compacts through the captured cut. XFS FICLONE creates an immutable standalone manifest; file and directory sync precede publication. A writable clone gets a private identity/log and a new COMMIT with $D = E = O = 0$ in its own sequence namespace. Catalog changes use temp-file, fsync, rename and directory fsync. Snapshot membership changes are excluded from host-wide quiescent GC; measure the pause.

1.12 Protocol [AFTER C5]

Message	Reply	Used by
GET(hashes)	bytes per hash	cold read, prefetch
PUT(segment)	ack after one fdatsync	compactor sending a sealed segment of chunks to an owner
HAS(hashes)	bitmap of hashes the owner lacks or has not fenced	compactor before PUT, so only missing chunks are sent; provisioning verification
LIVE(epoch, hashes)	ack	garbage collection
JOURNAL(image,ack after fdatsync range)		fleet class: the appends since the last FLUSH, sent to the journal peer

Messages are length-prefixed over kernel TCP with `TCP_NODELAY`, driven by `io_uring`. `GET` and `JOURNAL` have their own connections and priority. `PUT` is bulk. Every message is idempotent and named by hash or sequence number, so any of them can be retried. The daemon runs busy-polling or blocking. Page 04 measures both, because the scheduler wakeup is part of the cost.

1.13 Placement and the parameter k

Chunks are placed over N hosts by rendezvous hashing. Every host scores each (chunk, host) pair with one hash function, and the k highest-scoring hosts own the chunk. Every host computes the same owner set without shared state, a ring, or a lookup, at N hash evaluations per chunk. CRUSH's straw2 bucket is the same computation with per-host weights (NEED CITE). When a host joins or leaves, only the chunks whose top- k set changes move. The journal peer for fleet class is not chosen this way. A journal needs a fixed home with ordered replay, so each image names one peer at creation and keeps it. If a migration lands the guest on its own journal peer, the image names a new peer in the same fenced swap. On two hosts the journal peer is always the other host. k is the one multi-host parameter. With N hosts, $k = N$ places every chunk on every host (replicated) and $k = 1$ places each chunk on exactly one (partitioned). On the two-host testbed these are $k = 2$ and $k = 1$. Page 03 measures both, and a deployment would run $k \geq 2$ on $N \geq 3$ hosts.

1.14 Durability classes

Durability is a per-image class on one pipeline. The class changes who waits at FLUSH and for how long. Chunks reach the same owners either way; fleet class adds a copy of the staging tail at the journal peer until compaction catches up. **Local class**, the default: FLUSH returns after `fdatasync` of the staging log on this host. **Fleet class**: the appends since the last FLUSH are sent to the image's journal peer, which appends it to its own log and `fdatasyncs`. The send proceeds in parallel with this host's `fdatasync`, FLUSH returns after both, and FLUSHes from several images to the same peer share one round trip and one `fdatasync`. Fleet class is what Nutanix AOS and HPE SimpliVity do before they acknowledge, and page 04 measures what it costs.

Failure	Local class	Fleet class
daemon crash	completed writes remain readable; recover the log and inflight requests, then re-run compaction	same
host crash, power loss	FLUSH-covered writes survive; ordinary completed writes may be lost	same
host lost	the tail (O, E] can be lost; compacted chunks survive only if another host retains a copy	the staging tail survives: the journal peer replays (D, E] onto a new host; chunks the lost host owned survive only if $k \geq 2$, as in the row below
peer lost, $k = 1$	chunks it owned are unreadable until it returns, and lost if its disk is; a read that needs one waits or fails with an error, never returns stale bytes; writes use surplus copies while local capacity remains	the same for reads; a FLUSH waits for journal durability within its deadline or fails. The image does not silently change durability class

Reclamation preserves the selected durability class. In fleet class, the journal peer retains recovery data until durable chunks and recoverable mapping metadata on surviving hosts can replace it. A local surplus copy alone cannot release that remote journal data.

Two rules hold in both classes:

- The compactor never sends a chunk whose staging extent is not yet durable on this host.
- A transferred chunk replaces its local recovery copy only after the owner acknowledges durable storage and the manifest commit is durable.

1.15 Garbage collection (GC)

A chunk is live if any manifest on any host references it, or if an in-flight compaction has pinned it. A copy in a cache is never a reference. The baseline sweep pauses admission, drains IO and compaction, and marks active images and snapshots. Each host sends owners the complete live set for an epoch with `LIVE` before reclamation. Delete wholly dead store segments; copy live chunks from selected mixed segments into durable destinations before deleting the originals. Dead manifest pages can be hole-punched. Refcounting is not a concept in this architecture. ZFS frees an overwritten block when its reference count drops. This design does not, so space can leak between sweeps. The sweep therefore runs before every capacity measurement and when disk pressure requires reclamation. Report reclaimed bytes, copied live bytes and pause duration beside the capacity number.

1.16 Provenance

Component	Source	License
hypervisor	stock QEMU, unmodified, vhost-user-blk front end	GPL-2.0
vhost-user protocol	rust-vmm vhost-user-backend, vm-memory, virtio-queue; Cloud Hypervisor's vhost_user_block read as reference	Apache-2.0 / BSD-3-Clause
hashing	blake3 crate	CC0 / Apache-2.0
chunking	fastcdc crate	MIT
host filesystem	XFS on the dedicated NVMe, O_DIRECT, hole punching; ZFS never sits under the daemon	
staging, watermark, governor, compactor, store, index, manifests, cache, protocol, journal peer, garbage collection	this study	new code

2 Single host

The R0–R3 comparison runs the same stock QEMU, the same guest, and the same NVMe device, and varies only the storage behind the device. Hypothesis 1 predicts a tie on capture with ZFS fast dedup. The curve beneath the tie, capture against index memory across three chunk sizes, is this page's result.

2.1 Configurations

R0. Raw file on XFS. QEMU's raw driver on the dedicated NVMe. The control, with no deduplication anywhere in the path.

R1. Zvol on pinned OpenZFS fast dedup. Its own pool on the same device, created and destroyed per run, opened by QEMU as a block device. The current Nixpkgs lock selects OpenZFS 2.4.4. Freeze the release,

host kernel and pool settings for each measurement cohort; a version change reruns the affected controls.

Setting	Value	Why
feature@fast_dedup	enabled	DDT update log
dedup	blake3	SHA-256 with dedup=on
volblocksize	16K primary arm, 4K second arm	the zvol's dedup granularity
compression	zle	all-zero blocks become holes
dedup_table_quota	none	uncapped DDT
zpool ddtprune	never run during a measurement	no entries dropped
primarycache	all	explicit ARC policy
DDT memory	zpool status -DD, dedupcached	resident DDT bytes

OpenZFS 2.4.4 does not support its ARC-bypassing direct-IO path for zvols, so R1 remains ARC-backed. Filesystem direct writes instead skip deduplication; replacing the zvol with a direct-IO file would change the comparator. OpenZFS properties

The R1 profile verifies nonzero deduplication and zero handling before measurement. Record pool features, quota, sync settings, ARC residency and total host memory alongside the DDT measures. Report dedup_table_size as disk footprint and zero-run compression savings separately. Pool properties, DDT statistics

R2. Raw file on XFS over dm-vdo (optional). Fixed 4 KiB deduplication in the kernel, with its own XFS instance on the vdo device. Index memory from vdo stats.

R3. The backend on one host. Local store only, so k does not apply. Three chunk-size arms, below.

R0 against R3 is the cost of the daemon with everything else held constant. R1 is the deployed comparator and differs in kernel boundary, caching, and allocation, so it is a case study beside the controlled pair, and deltas are attributed accordingly.

2.2 Chunk-size arms

Fixed 4 KiB chunks cost one index entry per 4 KiB:

- about 250 million entries per TB
- about 10 GB of memory per TB at 40 bytes per entry, a 32-byte hash and an 8-byte offset

The alignment argument on page 00 predicts that they capture nearly every duplicate a Linux guest holds. The census measures the remainder. FastCDC with a 16 KiB mean cuts the index by four and loses an aligned 4 KiB match whenever the rest of its chunk differs. The one prior curve on VM images is Liquid's: 77% of bytes removed at 4 KiB, falling to 59% at 256 KiB on 183 images, with 256 KiB chosen for HDD seek cost. On NVMe the seek term is gone and the trade is index memory against capture.

Three arms: fixed 4 KiB, fixed 16 KiB, FastCDC 8 to 64 KiB with a 16 KiB mean. CDC boundaries snap to 4 KiB, so no guest block straddles two chunks and a 4 KiB overwrite invalidates one chunk, not two. Reported per arm: bytes stored, index bytes per TB, guest p99, write amplification, compactor CPU per GB. The census below predicts the capture column for each arm before any run.

2.3 Workloads

- fio: 4 KiB random write and read at QD1 and QD32; 128 KiB sequential.

- Boot storm: n clones of one image booted together, $n = 4, 16, 32$. A clone is a copy of the manifest with its own staging log.
- Fleet replay: the synthetic fleet below written onto n guests, at two points on its timeline.
- Overwrite: a small SQLite database rewriting its pages in place for an hour, with guest discard on.
- Pressure: sustained unique writes and overwrite bursts beside a reading guest; run through admission pressure and idle drain. Repeat reads with shared and disjoint working sets and a scanning neighbor.

2.4 Metrics

- Guest p50 and p99 write and read latency against R0, compactor active and idle. Reported first.
- Bytes stored after compaction completes and the sweep has run, against the census prediction at the configuration's chunk or block size. Bytes the sweep reclaimed reported beside it as the leak.
- Index or DDT bytes per stored TB.
- Write amplification: device bytes written per guest block-device byte, from NVMe counters. Record application bytes, virtio payload, staging records and fences, chunk writes, and metadata/GC traffic separately. Guest filesystem journaling can make application bytes differ from block-device bytes.
- Sustainable ingest, staging allocations, live and dead bytes, and compaction progress. Report the point where admission slows, each guest's latency, and idle drain time.
- Chunk traffic against the settle window: chunks produced per guest byte written, on the overwrite workload.
- Compactor CPU per GB ingested, per chunk-size arm.
- Host payload bytes copied, peak append/fetch-buffer bytes, cache occupancy and total resident memory, with shared mappings counted once.
- Recovery: the page 01 tests pass before any number is reported.

2.5 Controls

Pinned vCPUs, performance governor, discarded warm-up, fresh filesystem or pool per repetition, at least five repetitions, variance beside every number. With `cache=none`, R0 and R2 bypass the host file-data cache. R1 retains the ARC. Choose `zfs_arc_max` and the R3 clean-cache limit within the equal total memory budget below. Report actual ARC data/metadata residency, DDT residency, ZFS dirty memory and daemon buffers/indexes separately; equal cache caps do not establish equal total memory use.

All configurations are observed at the guest boundary (fio's histograms, guest-side blktrace for the boot storm) plus host device counters. The daemon adds per-request stage timestamps drained to ndjson, cross-checked once against bpfftrace with the delta reported. `zpool` and `vdostats` figures are supplementary.

Guest filesystem workloads retain normal caching. Direct guest fio runs isolate block-device costs and are labeled separately. Host IO mode, guest IO mode and FLUSH frequency are recorded independently. The host buffered/direct staging comparison holds format, queue depth, durability and total memory budget constant. Record packing and submission concurrency are varied separately. A one-block write followed by FLUSH exposes fence overhead.

Memory comparisons use equal total host budgets, including guest RAM, daemon buffers, indexes, caches and kernel file-data cache. Shared mappings are counted once. Slowed-compactor and owner-outage runs test the reserve and failure paths from page 01.

2.6 Guest memory sharing [PROPOSED]

A separate experiment would compare the same immutable image through virtio-blk and virtio-pmem/DAX, with identical private OverlayFS uppers. First verify DAX use, cross-guest isolation, and recovery of synchronized upper-layer writes. Then measure aggregate resident memory without

double-counting shared pages, guest memory use, read latency, CPU, and copy-up bytes as guest count grows. Hold contents, memory budgets, and workloads constant; separate cold host cache, warm host cache with cold guest cache, and repeated accesses.

This changes the guest storage stack, so it has its own results table alongside R0–R3. Its first question is whether mapped reads reduce memory duplication for shared files. No performance threshold is set; further CAS integration depends on the feasibility result.

2.7 Payload placement [PROPOSED]

The baseline copies surviving staging data into a separate chunk store. An alternative at fixed 4 KiB would hash surviving records and publish their existing payload locations. Segment cleaning could still move live data to reclaim space. Larger chunks assembled from scattered writes may require copying.

Both layouts must recover acknowledged FLUSHes before comparison. Hold workload, durability, memory and available disk space constant. Measure unique writes, overwrites and deletion at increasing disk occupancy; report all payload writes, retained segment bytes, cleaning traffic and guest latency. This experiment leaves the guest contract unchanged.

2.8 Census

A small census supplies the numbers the rest of the study is measured against: how many unique bytes the fleet holds under each arm's chunker, and how many bytes copy-on-write would already have shared.

Phase 0. `zdb -S` on a ZFS pool holding the cloned fleet. Pool traversal starts each dataset at its origin snapshot's transaction group, so blocks a clone inherited are counted once and the simulated ratio is duplicates beyond what clones already share. This reading of `dmu_traverse.c` is confirmed with a two-clone test before the number is cited.

The fleet. Ubuntu publishes dated cloud images and a dated package archive. The controlled Ubuntu fleet uses that archive; Debian's `snapshot.debian.org` is an alternative for Debian guests. An image installed as of T0 and upgraded monthly against the archive as of T1, T2, and on replays a real update history. `n` such clones with scripted drift (hostnames, logs, a few packages each) form the fleet. It is rebuilt by one command, dated, and is also the replay workload above.

The split. Per byte range: zero or unallocated (from the guest allocation map, excluded), unique, shared with the T0 base in place, duplicate at an aligned 4 KiB or 16 KiB boundary elsewhere in the fleet, or duplicate only at a shifted offset. The aligned columns predict R1 and the fixed arms. The CDC arm is predicted by running FastCDC with the arm's parameters over the images, because a 16 KiB mean chunk captures fewer aligned matches than fixed 4 KiB chunks and more shifted ones, and the two effects do not add. There are no donors, no real fleets, and no claims about time.

3 Multiple hosts

Page 03 runs the backend on two hosts with one parameter, `k`, and measures bytes moved and bytes stored against what `zfs send, rsync`, and two per-host ZFS pools would move and hold.

3.1 Two placement modes

k , the number of owners per chunk (page 01), takes two values on two hosts, and they are two different experiments. Surviving a dark host at two hosts costs a full mirror of chunks ($k = 2$) plus fleet class for the staging tail.

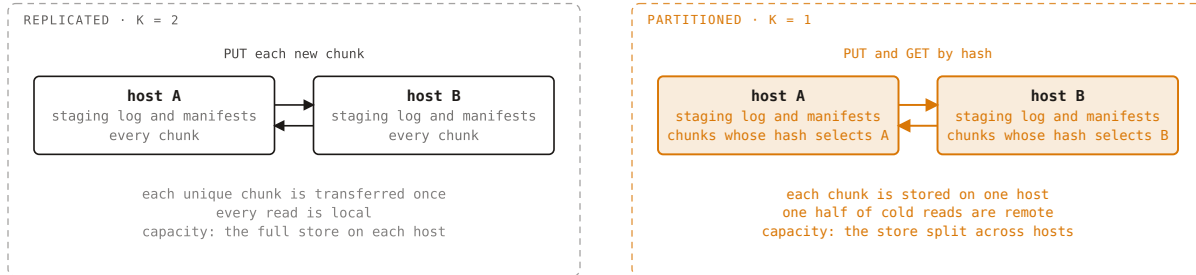


Figure 6: $k = 2$ provides transfer savings and keeps every read local. $k = 1$ provides capacity savings at the cost of remote reads. Two hosts with $k = 1$ send one half of cold reads to the peer in expectation, the largest share this testbed can produce.

3.2 Provisioning

A new guest on host B from an image whose chunks exist anywhere costs a copy of the manifest: at least 32 bytes per chunk, about 80 MB for a 40 GB image at 16 KiB chunks. Every chunk it names already exists at its owner. In replicated mode no other data is transferred. In partitioned mode no other data is transferred either, because chunks are fetched on first read.

The baseline is `qemu-img convert` or `scp` of the raw file and `zfs send | zfs recv` of the zvol, each moving the allocated size of the image. Liquid cloned an image by copying its metadata file, in milliseconds. Its distribution benchmark moved an 8 GB image to seven nodes on 1 GbE in 35 s against 730 s by `scp`, and still moved every unique block to every node. Here provisioning is bytes on the wire at 100 GbE.

3.3 Migration

To move a guest from A to B, the daemon freezes the device on A and takes E, hands the image to B by one fenced swap of its root record, ships the manifest and the staging extents in (D, E], and resumes on B. The root record names the writer and carries a generation number, and the swap is written durably on both hosts before B resumes. A accepts no write after the swap, and B resumes only after the swap names it. On resume the log is reconciled by evidence, the replayed E against what is durable on disk, never by who claims to own it. In a prior implementation by the author, a refusal keyed on writer identity kept healthy guests from restarting. A 40 GB image that compacted recently moves its manifest, about 80 MB at 16 KiB chunks, plus the staging tail, which was under 9 MB for an idle guest in that implementation and is workload-bound for a busy one.

We predict that bytes are the small part of a migration. The disk cut measured 3 to 6 ms in that implementation and the rest of the blackout was orchestration, so the blackout is reported decomposed into freeze, swap, transfer, and resume, beside the bytes. Governor pacing is disabled while the guest is paused.

The baseline is `rsync` of the raw file and `zfs send` of the zvol, which since 2.0 emits no deduplicated stream (page 00), so both move the allocated size.

3.4 Synchronization after drift

Two guests, one on each host, are cloned from the same image and each updated independently to the same package set. Compaction on each host sends only the chunks the owner lacks, packed in sealed segments. Bytes on the wire are read against the census's unique-byte count for the pair. Chunks per second is reported beside bytes per second, the compactor's ship rate, because we predict per-chunk cost caps the path before the link does. This is the apt upgrade case from page 00, measured.

3.5 Capacity

Partitioned mode stores each chunk once across the fleet. Bytes on both stores after the fleet replay and the sweep are measured against two per-host ZFS pools holding the same guests, with index bytes on each host. Hypothesis 2 predicts at most 55% of the pools' bytes and about half the index per host. The fraction of a guest's cold reads served by the other host is measured too. On two hosts with $k = 1$ that fraction is one half in expectation. In general it is $1 - k/N$, so a larger fleet at fixed k sends a larger share of its cold reads over the network, and the two-host number is a lower bound on that share.

3.6 The local-class window

In local class, between a FLUSH acknowledgment and the chunk being durable at its owner sits the compaction lag, $(O, E]$ in the watermark's terms. It is reported in seconds under the fleet replay, as a distribution, with the segment size as the parameter. This window measures owner-confirmation lag. Host-loss recovery also requires surviving chunk copies and mapping metadata, as specified on page 01.

Fleet class protects the staging tail through the journal peer and pays one round trip and one remote fdatsync per FLUSH. Page 04 measures that cost.

3.7 Measurements

Flow	Daemon	Baseline	Read against
provision	bytes transferred, both modes	scp of raw file; zfs send	manifest size
migrate	bytes transferred, both modes; blackout decomposed	rsync; zfs send	manifest size + staging tail; milliseconds for the cut
sync after drift	bytes and chunks per second sent by compaction	rsync; zfs send	census unique bytes
capacity	bytes stored, partitioned, after the sweep	two per-host ZFS pools	at most 55% of the pools' bytes (hypothesis 2); census prediction
index per host	index bytes on each host, both modes	DDT bytes per pool	k/N of the fleet index
remote fraction	cold reads served by the peer		one half in expectation
local-class window	seconds from ack to owner-durable		owner-confirmation lag

3.8 The locality objection

Dong et al. (page 06) rejected per-chunk hash placement for backup streams on locality grounds. This is primary storage with a local cache, so page 04 measures the fragmentation cost directly, and super-chunk placement is the knob if it is large.

4 Remote read

Page 04 measures the one place the network enters guest latency in local class, a cold read whose chunk lives on another host, then how much of it prefetch removes, then the round trip fleet class adds to FLUSH.

4.1 Where the time goes in a remote read

A 4 KiB random read at QD1 from an enterprise NVMe SSD completes in about 80 μ s: 81.6 μ s on a PM1725 in Systor '17 and about 80 μ s on a PM1735 in blk-switch. R0 measures the testbed's drive in week 1. On 100 GbE the transport sits on top. Against a null device, kernel nvme-rdma added 12.1 μ s and kernel nvme-tcp 21.4 μ s for a 4 KiB read on ConnectX-5 (SPDK 24.05). A raw RDMA round trip is 2 to 5 μ s, from 32 bytes to 4 KiB (eRPC; SPDK). On two CloudLab c6525-100g nodes, the testbed's node type, BPF-oF measured average round trips of 18 μ s over nvme-rdma and 30 μ s over nvme-tcp on kernel 5.12. A userspace daemon over kernel TCP has no published measurement as a remote read target. From the kernel TCP round-trip floor of 13 to 23 μ s (Homa; Zuo et al.) plus a file read, we estimate 20 to 30 μ s when it polls and more when it sleeps. The testbed replaces every one of these figures.

On these figures the difference between RDMA and TCP is about 9 μ s on a read of about 100 μ s. The larger factor, about 4x, is whether the chunk is in the owner's memory or on its disk. **If the figures hold, a chunk from a peer's memory over TCP arrives before one from local NVMe**, and hypothesis 3 tests this. Every host sends its reads of a chunk to the same k owners, so we predict a chunk read by many guests is hot at its owner, and a remote read in that case is the memory row. The owner's hit rate is reported under the boot storm below.

A caution from a prior implementation by the author: a peer round trip over QUIC with TLS on a bonded 25 GbE link measured 108 μ s at p50 and 257 μ s at p99 (unpublished). The daemon here uses kernel TCP with TCP_NODELAY on 100 GbE.

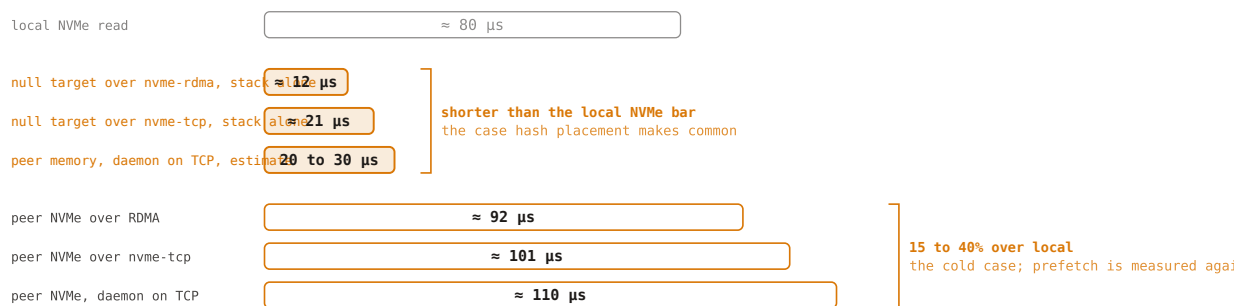


Figure 7: Literature values for one 4 KiB read at QD1, in microseconds, before the testbed measures them. The daemon rows are estimates.

4.2 Transport probes

The architecture's transport is the daemon over kernel TCP. The other rows exist to show what the kernel stack and the userspace hop each cost.

Probe	What it isolates	Code
ib_read_lat -s 4096	the hardware floor	none
nvme-rdma export	kernel block path over RDMA; owner's store exported by nvmet as a file-backed namespace, buffered_io on for memory, off for media	configuration
nvme-tcp export	same over kernel TCP	configuration
daemon, TCP, busy-polling	the architecture, without the wakeup	the daemon
daemon, TCP, blocking	the architecture as deployed; the scheduler wakeup is the cost	the daemon
daemon, ibverbs two-sided[STRETCH]	the userspace hop without the kernel stack	~40 h

The nvmet export is a probe and not the architecture. It exposes the raw store, needs the reader to know offsets, and has no place for authentication. It is in the table because the difference between it and the daemon over the same TCP is the cost of the userspace hop, with the delta between SPDK's userspace target and the kernel's, 1 μ s on TCP and 2.7 μ s on RDMA in the 24.05 reports, as the reference point.

4.3 Method

- Same two hosts, NIC, drive, and kernel for every row. Kernel, firmware, MTU, IRQ affinity, interrupt moderation, C-states, busy-poll, and PFC state recorded.
- The link is measured before any remote number: ib_read_lat for the RDMA floor and a TCP ping-pong for the kernel floor, both recorded beside the rows.
- Two targets per row: a null device for fabric plus stack alone, and the real file for end to end. Each from the owner's memory and from its NVMe.
- Two load states for the file rows: quiet, and with PUT traffic running on its own connection at the ship rate from page 03, because a cold read in deployment competes with compaction. The difference is what the read-priority rule on page 01 buys.
- 4 KiB, 16 KiB, 64 KiB. p50, p99, p99.9. Five runs of 30 s, caches dropped between, medians with spread.
- QD sweep 1, 4, 16, 64 for throughput and CPU per IOPS on both ends. Kernel TCP costs 2.5 to 3x the CPU of RDMA at equal IOPS in the SPDK 24.05 reports and in i10, and the ratio measured here is reported.
- RoCE hardware counters (out_of_sequence, packet_seq_err, local_ack_timeout_err) printed beside every RDMA number, showing zero retransmits on a fabric with no PFC.

4.4 Prefetch

The depth sweep reads sequentially through the manifest with P chunks in flight, P doubling from 1 to 512 at 4 KiB and from 1 to 32 at 64 KiB. The bandwidth-delay point is about 250 KB for the fabric (100 Gb/s $\times$ 20 μ s) and about 1.2 MB with media under it (100 Gb/s $\times$ 100 μ s), so we predict that about 20 chunks of 64 KiB or 300 of 4 KiB in flight hide the remote read. Success is remote sequential throughput within 10% of local, the throughput clause of hypothesis 3.

Profile prefetch records the chunk sequence of one boot and replays it on later boots. DADI, REAP, FaaSnap, VMTorrent, and Nydus each prefetch a recorded access profile (page 06). It is budgeted at one day.

4.5 Under a guest workload

Partitioned boot storm at $n = 16$, with and without profile prefetch, against the same storm in replicated mode. The run reports guest p99, host device reads per guest byte, the fraction of reads served by the peer, and the fraction of those the peer answered from memory, so the per-read cost and the miss rate can be multiplied and the hotness prediction on page 00 is checked. **The gap between partitioned with prefetch and replicated is the residual cost of one copy per chunk.**

4.6 The FLUSH round trip

Fleet class puts one round trip and one remote fdatsync in front of every FLUSH acknowledgment. On the figures above the transport is about a tenth of a cold read over RDMA and a fifth over TCP, because $80\ \mu\text{s}$ of media sits beneath the read. A FLUSH has no media to hide behind, so we predict the round trip and the peer's fdatsync are most of its cost, and hypothesis 4 bounds it. It is measured here with the same discipline as the read rows: write p99 at QD1 for local class, for fleet class over the daemon on TCP, and for fleet class over ibverbs if that arm lands, with the peer's fdatsync time reported separately so the transport's share is visible.

4.7 RDMA on this testbed

The CloudLab fabric is lossy. No PFC or ECN is documented on the shared switches, and the one published RoCE measurement on this node type does not say whether either was on. Adaptive retransmission is enabled on the NIC and the counters above show whether the runs were clean. ConnectX-5 cannot do io_uring zero-copy receive, so that option is unavailable.

5 Plan

Fourteen weeks at about 320 hours, which is 23 a week against the 8 the course credit corresponds to. The plan is sized to the work, and the descoping order defines what is removed if it slips.

5.1 Development checkpoints and research gates

The schedule below is the research plan. The architecture page records the dated C0–C5 implementation status; TODO.md is the canonical checklist. C0–C3 are accepted, the C4 functional inventory passes, and C5 has validated shared guests, caches and integrated scheduling. Pressure workloads and final memory accounting remain open.

Development checks run on Spark with disposable KVM/XFS and nested filesystem guests. G1–G6 and dedicated-media repetitions remain pending under their own acceptance conditions. Each session retains concise results and failures; large storage artifacts follow the archive and disk-headroom policy.

5.2 Hardware

The planned measurement testbed is two CloudLab c6525-100g nodes (Utah), to be reserved as a pair. Each node has an AMD EPYC 7402P with 24 cores at 2.80 GHz, 128 GB ECC DDR4-3200, two 1.6 TB PCIe 4.0 NVMe SSDs, and a ConnectX-5 Ex 100 GbE with one port on the experiment network. One NVMe holds the system and results, and the other is the device under test. The pair is one hop through a single switch.

RoCE between two of these nodes works on the lossy fabric, since BPF-oF ran it there (page 04). The testbed uses the repository's pinned NixOS host configuration. R1 adds a compatible kernel and the pinned Open-ZFS userspace/module, built and boot-verified together before measurement. The host closure, locks and runtime versions are archived per cohort. The dedicated pair and its installation path remain pending. An experiment expires after a few hours unless it is extended, so every run is scripted to complete inside one sitting.

CloudLab is free for research. A project is opened by a faculty member and reviewed by CloudLab staff, so the sponsor opens it before Sep 9. The fallback is two OVHcloud Advance-4 2026 servers (EPYC 4585PX, 16 cores, 64 GB DDR5 ECC, 2 × 960 GB NVMe) on a 25 Gbps private link, which loses the RDMA arm and replaces the 100 GbE fabric with 25 GbE.

5.3 Schedule

Weeks	Build	Measure
1–2	vhost-user-blk daemon in passthrough: staging log, FLUSH, replay. Kernel and ZFS image.	R0, with the drive's read and fdatsync times; passthrough within 10% of R0 p99 (G1). Thresholds frozen. zdb -S phase 0 on the synthetic fleet.
3–5	Compactor with settle window, store, index, manifests, watermark, governor, recovery. Three chunk-size arms.	kill -9 recovery and the three ordering tests pass (G2). First capture numbers.
6–7	R1 configured, both volblocksize arms. R2 if time permits.	Page 02 table complete (G3), sweep before every capacity number.
8–9	Protocol with separate GET and PUT connections, rendezvous placement, k, segment PUT with durable ack, HAS, pins, surplus copies, sweep. Provisioning; migration with the fenced handoff.	Replicated mode on two nodes.
10	Partitioned mode. Fleet class over TCP.	Page 03 table complete (G4).
11–12	nvmet exports, RoCE configuration, busy-polling and blocking daemon, depth prefetch, profile prefetch.	Transport matrix and prefetch sweeps (G5). Partitioned boot storm.
13–14		Report; reproducibility pack (G6).

5.4 Gates

G1. Passthrough daemon under stock QEMU within 10% of R0 p99 by the end of week 2. If this slips, everything after it slips, and the sponsor is informed that week.

G2. The recovery and ordering tests on page 01 pass before any daemon number is reported.

G3. Page 02 table complete: R0, R1 at two block sizes, R3 at three chunk sizes; latency, capture, index, amplification; variance beside every number.

G4. Page 03 table complete: both modes, every flow, each read against the bound its row names.

G5. Transport matrix complete for every non-stretch probe, null and file, memory and NVMe, with the RoCE counters printed beside every RDMA number.

G6. One command rebuilds the fleet from dated archives, and one command reruns every table on a fresh

pair.

5.5 Descoping order

When the schedule slips, items come off from the top.

1. ibverbs daemon arm, and with it fleet class over RDMA.
2. Super-chunk placement.
3. R2 dm-vdo.
4. Profile prefetch (depth prefetch stays), and with it the boot-storm clause of hypothesis 3.
5. Fleet class over TCP. Hypothesis 4 is then reported as untested, with the literature's numbers as the estimate.
6. Partitioned mode. Replicated mode alone still gives hypothesis 2's transfer result, and the remote read of hypothesis 3 is then measured with the local copy disabled so that the read is forced to the peer.

Not removed under any slip: page 02, the nvmet TCP probe, and the daemon over TCP. The RDMA probes go if RoCE configuration exceeds its budget or the fallback hardware is used.

5.6 Risks

- **Daemon overrun.** The largest risk and the reason G1 is at week 2. Protocol plumbing comes from maintained crates, so the hours go to the components listed as new code on page 01.
- **RoCE configuration.** GID selection, MTU, adaptive retransmission on a lossy fabric. Budgeted at 8 hours. If it exceeds 20, the RDMA rows are dropped and the TCP rows stand.
- **Node availability.** 36 nodes of this type exist, so the pair is reserved in week 1 for every measurement week.
- **Correctness debt.** The defects that stall or corrupt a guest are known from a prior implementation, and each has a test on page 01 and hours in weeks 3 to 5, before any number is taken.
- **O_DIRECT alignment.** Final append buffers satisfy the backing filesystem's alignment requirements. Verify addresses, offsets and lengths independently of the guest block size; report payload copies and any buffered fallback.
- **Known configuration pitfalls.** The 100G interface stays down unless the profile declares a link on it; the ZFS pitfalls are in the R1 table on page 02.
- **Census realism.** Scripted drift is not real drift. The fleet is built from real dated archives, the scripts are published, and the numbers it supplies are bounds the daemon is read against, not claims about fleets in the wild.

5.7 Logistics

CS 4993, 1 credit. Expectations in writing before Sep 9. Thirty minutes of sponsor time every two weeks, with G1 as a scheduled meeting.

5.8 Future work

Availability. Fleet class is the seed of replication before acknowledgment. With it and $k \geq 2$ on $N \geq 3$ the system has a failure model, which needs membership, failure detection, and rebalancing, none of which this study touches.

Placement and reclamation. Super-chunk placement for locality. A cache policy that weighs a chunk's owner distance. An on-disk copy-on-read tier for chunks that are cold at their owner. Reference counts kept

as derived state, with the sweep as the auditor, so an overwrite frees space at once as it does in ZFS.

The same split elsewhere. Prefix caching in LLM serving (vLLM, SGLang, Mooncake) names cached KV blocks by a hash chain over the whole token history, so two requests share only along a common prefix. That is lineage. The same document after two different preambles is computed twice. That is the cross-host case here, and its size on a real trace is unmeasured.

6 Prior art

Swept on 2026-09-01. Sources and what was opened are in docs/review/. No system in the sweep combines a durable, sequence-numbered local write log with a stated FLUSH contract, a fleet-wide chunk store whose owners are the hosts themselves, a block device under a stock hypervisor, and a per-transport measurement of the remote cold read. Liquid (TPDS '14) is the nearest design and the row to read first. Datrium, Nutanix, and Fossil with Venti each share one component.

6.1 Nearest systems

Work	What it is	How this differs
Datrium DVX (2016), US20170031994A1	host-side fingerprinting, host flash as read cache, global deduplication on a shared data-node pool; the patent lists host-only ack as an alternative	peers as owners by hash instead of a shared pool; open implementation on stock QEMU; the cold read measured per transport
Nutanix AOS	local OpLog on SSD, mirrored to another node before ack; cluster-wide post-process deduplication at 16 KiB; per-node cache	no mirror on the write path by default, with the window measured and the mirror as fleet class; placement by hash instead of by vDisk locality; latency and capacity numbers, which its documentation does not give
Fossil + Venti (2002)	a disk write buffer in front of a content-addressed archive; the two-tier shape	block device under a VM instead of a filesystem; primary capacity instead of archival; more than one owner
Ceph + TiDedup (ATC '23)	post-process CDC into a chunk pool placed by CRUSH on the fingerprint; promotes on a cold miss	guest writes are acknowledged before any byte crosses the network; a host cache instead of promotion; a guest block path; latency numbers, which TiDedup does not report
vSAN ESA global deduplication (2025)	cluster-wide post-process 4 KiB deduplication, mirrored writes, 3 to 16 hosts, no published numbers	the per-host to cluster-wide change this study measures, with published numbers
HYDRAsTOR (FAST '09)	content-addressed blocks placed by DHT across a grid, global deduplication	secondary storage with network writes; no guest path
DeDe (ATC '09)	hosts hash in-band, deduplicate out-of-band against a shared index on a SAN, no coordinator	local disks instead of a SAN; chunks move to owners instead of pointers on shared storage

Work	What it is	How this differs
Liquid (TPDS '14)	FUSE file under a stock hypervisor; fixed 256 KiB to 1 MiB blocks hashed on flush or eviction from a 256 MB volatile write cache, pushed to range-partitioned data servers at VM shutdown; central meta server with refcounts; P2P Bloom-filter cache tier; copy-on-read disk cache; two replicas	a durable log with a FLUSH contract instead of a volatile buffer with no crash story; a vhost-user block device instead of FUSE; hosts as owners by rendezvous instead of a meta server and a data-server tier; exact HAS instead of Bloom filters; the miss cost measured, which Liquid names ("several times longer") and never measures

6.2 Remote fetch in prior systems

Work	What it measured	What it leaves open
Liquid (TPDS '14)	8 GB image to 7 nodes on 1 GbE: scp 730 s, NFS 510 s, BitTorrent 95 s, Liquid 35 s; on-demand boot 1.7x to 4x a cached boot	miss cost stated as "several times longer IO delay" and never measured; no latency numbers anywhere; HDD and 1 GbE
DADI (ATC '20)	block-level lazy loading with tree P2P; 10,000 containers on 1,000 hosts in 4 s; trace prefetch removes 95% of the cold gap; reads from a parent's page cache are faster than local disk	no per-read miss latency; not content-addressed
Slacker (FAST '16)	only 6.4% of a container image is read at startup; lazy fetch over NFS; run phase 17% slower	no per-block miss cost; centralized
VMTorrent (CoNEXT '12), VMThunder (TPDS '14)	demand-priority P2P VM image streaming; VMTorrent replays recorded profiles, VMThunder streams on demand through a relay tree	startup seconds only
FaaSnap (EuroSys '22), REAP (ASPLOS '21)	lazy page faults from local disk at 13 μ s; userfaultfd over 128 μ s uncached; working set 9% of footprint	memory, not disk; local
SnowFlock (EuroSys '09)	275 μ s per page fetched over gigabit, 82% of it in the network stack	the only in-VM remote per-unit number, and it is from 2009
Dahlin et al. (OSDI '94)	cooperative caching: remote client memory at 1.25 ms against disk at 15 ms; N-chance forwarding	the argument this study repeats at 100 GbE with content-addressed chunks
CLB (VEE '17), Satori (ATC '09)	content-keyed sharing of VM disk reads across guests on one host; 94.9 to 98.5% of boot reads eliminated	single host; no store

Among the systems above, which span FAST, ATC, OSDI, EuroSys, ASPLOS, CoNEXT, VEE, and TPDS from 1994 to 2022, none reports the latency of a content-addressed chunk fetched from a peer inside a VM block read path. DADI, VMTorrent, REAP, and FaaSnap report startup time in seconds and hide the per-read penalty behind a recorded access profile, and VMThunder reports seconds without one. Liquid

names the penalty and does not measure it.

6.3 Objections already in print

Dong et al. (FAST '11) rejected per-chunk hash placement for backup streams on locality grounds and routed 1 MB super-chunks. Page 03 answers with a local cache and page 04 measures the cost. **Meyer and Bolosky** (FAST '11) found that deduplication savings grow with the log of the number of file systems in one domain, on 857 desktops. Two hosts is the floor of the capacity half of hypothesis 2, and a larger fleet gains more. **despairlabs** (2024) tells ZFS operators to use clones and block cloning for the copy case and deduplication rarely. Hypothesis 1 predicts agreement on one host, and hypothesis 2 measures the cross-host case that advice does not address. **The hyperconverged products** (Nutanix, Datrium, SimpliVity, vSAN ESA) mirror a write over the network before acknowledging it, so a local-only acknowledgment is a durability trade rather than a free latency gain. Page 01 makes it a class and page 04 prices it.

6.4 What this study adds

The study's contribution is the measurement: what content addressing provides across hosts on commodity hardware under a stock hypervisor, and what the remote cold read costs, per transport.