p4-cache

Tiered storage for Perforce.

Production-grade. Audited. Transparent.

Cut Perforce storage costs by 60–80%.

Without changing a single developer workflow.

The problem

Your depot grows. Premium storage doesn't get cheaper.

Every Perforce admin running a depot above a few terabytes knows the storage cost curve. Active projects expand. Historical projects accumulate. Sunsetted titles never quite get archived. The whole depot sits on enterprise SAN or premium cloud SSD because that's what `p4d` needs to serve `p4 sync` at speed — and most of it hasn't been read in months.

The standard workarounds all cost something:

- **`+X` archive filetypes, managed by hand** push the work onto the admin: per-class decisions, manual restoration, a second depot to manage. The `+X` hook itself is sound — p4-cache automates it end-to-end rather than leaving it as manual toil.
- **`p4 obliterate`** trades storage cost for irreversible deletion. Most admins won't run it without a signed-off retention review, and the time to get one usually exceeds the cost of just buying more disk.
- **NetApp / Pure / Isilon refresh.** Every 4–5 years, write another seven-figure check for the same architecture that's been getting slower-relative-to-the-depot for a decade.
- **Bigger cloud premium SSD.** Scales linearly with depot growth at $900–1,500/TB/year. Painful at 30 TB; brutal at 300 TB.
- **Hand-rolled blobfuse2 or s3fs setups.** Work until the engineer who built them leaves, the manifest gets out of sync with the bucket, or a quiet content-hash mismatch corrupts a sync six months from now.

Why this gets harder every quarter

5–15% of any Perforce depot is read in any 30-day window. The other 85–95% sits on premium storage because some of it might be needed quickly, and `p4d` doesn't distinguish between blocks hit constantly and blocks not touched since 2019. The depot is uniformly priced: paying the rate of the worst-case access pattern, applied to the entire depot.

This is the cost gap that p4-cache exploits.

The product

How p4-cache works

p4-cache is a long-resident daemon plus a per-call `p4cache-trigger` binary that p4d invokes via its `+X` archive trigger for every archive op on `+X`-tagged depot files. Together they tier cold depot files to your chosen backend and rehydrate them on demand — transparently to `p4d`.

At read time. When p4d serves a `+X` archive it isn't holding locally, it spawns `p4cache-trigger read <file> <rev>`. The trigger speaks a versioned binary (postcard) protocol over a local Unix socket (Linux abstract address) or named pipe (Windows) to the daemon. The daemon serves from its local NVMe cache when present; on miss it restores from the configured object store, streaming-MD5-verifies the bytes against `p4d`'s own `db.storage.digest` before letting them reach p4d, and hands the trigger a file descriptor via `SCM_RIGHTS` for zero-copy delivery. Local cache hits return in ~1 ms warm.

At write time. p4d spawns `p4cache-trigger write <file> <rev>` and streams the new archive bytes to its stdin. The daemon writes them atomically to the mirrored archive path on local NVMe, marks the upload manifest `Dirty`, and returns immediately — `p4d`'s submit never waits on the cold tier. An async worker pool drains the Dirty queue in the background (Dirty → Uploading → Clean). On daemon restart, any `Uploading` rows revert to `Dirty` so a fresh worker re-tries; uploads are idempotent against the bucket key.

Source-of-truth split. p4-cache does NOT maintain a parallel `(depot, rev) → file` index. The on-disk archive tree IS the index (`<archive_root>/<rel>,d/<lbrRev>(.gz)?`). Refcount, digest, and lbrType come from p4d's `db.storage` (already replicated by SDP) via `p4 -ztag storage`. The daemon's redb manifest tracks ONLY in-flight upload state.

Hot tier: bring your own NVMe

p4-cache doesn't bundle hardware and doesn't care where the NVMe comes from:

- **Existing servers** with NVMe you already have. Point p4-cache at the local mount.
- **New commodity hardware.** A 2U server with 100+ TB of enterprise NVMe runs $20–40K capex. Five-year fully-loaded TCO lands well under $200/TB/year.
- **Cloud NVMe instances** (Azure Lsv3, AWS i4i, GCP n2-highmem with local SSD). Supported.

Cold tier: any of the major options

Every cold-tier backend dispatches through one code path — the Apache Arrow `object_store` crate.

- **AWS S3** — static credentials or SDK default chain. S3-compatible services (MinIO, Wasabi) via endpoint pinning.
- **Azure Blob Storage** — SAS, SharedKey, or Azure SDK default credential chain (managed identity, CLI login, environment).
- **Google Cloud Storage** — service account JSON or Application Default Credentials.

- **NFS / local filesystem** — any mounted POSIX path, for hybrid deployments or shops with existing on-prem secondary storage.

Two ways customers buy p4-cache

Same product. Different framing. Different commercial structure.

Cloud studio

5–50 TB depot on cloud Perforce. Pain is the monthly cloud bill, asked about quarterly. Typical buyer: Perforce admin or DevOps lead with VP Engineering approval.

Typical 80 TB deployment — annual cost breakdown:

	Current (Azure Premium)	With p4-cache
Storage platform	$192,000	—
p4-cache license	—	$16,000
Hot NVMe tier	—	$0 *(repurpose existing)*
Cold tier (200 TB blob)	—	$24,000
Annual total	**$192,000**	**$40,000**
Annual savings		**$152,000 (79%)**

License pays back in under 2 months.

Enterprise displacement

200 TB – 2+ PB depot on NetApp / Pure / Isilon. Pain is the refresh quote on the CIO's desk. Typical buyer: storage architect with Director / VP Infrastructure sponsorship.

Typical 1 PB deployment — annual cost breakdown:

	Current (NetApp)	With p4-cache
Storage platform	$4,500,000	—
p4-cache license	—	$100,000
Hot NVMe tier (120 TB commodity)	—	$45,000
Cold tier (2,380 TB blob)	—	$690,000
Annual total	**$4,500,000**	**$835,000**
Annual savings		**$3,665,000 (81%)**

Plus avoided refresh CapEx — typically $2–3M every 4–5 years.

Five-year picture

For a multi-petabyte Perforce shop facing a refresh decision, p4-cache changes the math entirely:

Path	5-year TCO	What it requires
Refresh the array	**$7–15M**	*Status quo CapEx pulse*
FabricPool to S3	**$5–9M**	*Stay on NetApp forever*
p4-cache + commodity NVMe + blob	**$4–5M**	*New license, no refresh*

Production-grade, with citations

This is the pillar most year-1 commercial products cannot claim. p4-cache can.

Engineering maturity

- **~40K lines of Rust across six workspace crates** (`p4cache` daemon, `p4cache-trigger` archive trigger, `shared` types/codec, `license` envelope verification, `license-tool` vendor signer, `integrity-stamp` artifact stamper). 400+ tests across the workspace.
- **Three independent code audits** completed: quality, security, and verify-binary verification. All findings tracked in repo `*-REMEDIATION.md` files.
- **Zero open CRITICAL findings** today, where CRITICAL is defined as data loss, RCE, auth bypass, secret leak, persisted state corruption, or known-exploit CVE in a direct dependency. **Zero open HIGH engineering findings.** Two remaining HIGH supply-chain advisories are transitive deps pinned by upstream SDKs (waiting on upstream bumps; ignored in `rust/.cargo/audit.toml` with documented rationale).
- **Compile-time defenses** against entire classes of bugs (Rust edition 2024): every unsafe operation requires an explicit unsafe block; holding mutex guards across await boundaries is a compile error; silent integer casts produce warnings.
- **CI gates** on `cargo audit` and `cargo deny` (advisories, bans, licenses, sources).
- **Fuzz harnesses** on the trigger request handler and the Perforce checkpoint parser. Run in CI smoke mode on every push.

Defenses by category

Process isolation. The archive trigger is a per-call static binary spawned by p4d — a clean process boundary, with no in-process libc hooks loaded into p4d itself. A trigger crash exits non-zero and surfaces as a stock p4d archive-op failure; it cannot affect p4d's process state. The daemon's per-connection trigger handler is wrapped in `catch_unwind` so a panic during a single request becomes a structured ERROR log rather than killing the listener.

Authentication boundaries. Trigger-socket access on Unix is gated by `SO_PEERCRED` against the configured `trigger_allowed_uids` allowlist. Depot path and revision strings are validated against traversal, NUL, LF/CR, oversize, and `-`-prefix argv tricks before any handler reaches the filesystem, the bucket key, or `p4` argv. All rejections are logged.

Data integrity. Every cold restore is streaming-MD5 verified against `db.storage.digest` — the same hash p4d already stores. Bytes are written into a staging temp file and only atomic-renamed onto the live archive path after the hash check passes. The cache write path is atomic (temp + fsync + rename + fsync(parent)). The cache layout mirrors p4d's archive layout exactly (verified by `rust/tests/p4_gzip_parity.sh`), so a daemon outage degrades to "p4d sees the file at the path it expected" rather than corruption.

Forensic watermark. Every cold-tier write carries a deployment-identifying watermark in object metadata (`p4cache-customer-id`, `p4cache-license-id`, `p4cache-daemon-instance`, `p4cache-source`, `p4cache-tier`) and, on S3/Azure, in the cloud SDK `User-Agent` suffix. A leaked or exfiltrated blob is traceable to the licensed deployment that produced it. Capacity and expiry states surface as Prometheus metrics for operator alerting.

Audit trail. Every depot file read is recorded to Elasticsearch or PostgreSQL with TLS (custom CA supported), deduplication, batching, and optional on-disk spool for outage resilience. Two distinct drop boundaries (pre-spool queue overflow vs. post-spool spool overflow), both observable in Prometheus. No silent drops. The license audit log is a tamper-evident NDJSON chain with cross-restart anchor (`license-audit.anchor.json`).

> ***What this means for your security review***
>
> *The Security & Compliance Brief is the foundational document for an enterprise security review. It cites specific ADRs and audit findings, documents the threat model, and enumerates defenses by category. Most year-1 products require weeks of security-team investigation before approval. p4-cache provides the answers up front. Available under NDA.*

Pricing & next steps

Pricing

Per-TB managed capacity. No per-seat licensing. No feature gating. Free 60-day evaluation — feature-complete, capacity-limited (500 GB hot / 5 TB cold).

Depot size	Annual list	Buyer profile
5 TB	**$2,000**	*Admin / lead approval*
30 TB	**$6,000**	*VP Engineering signoff*
100 TB	**$20,000**	*Director / VP Infrastructure*
500 TB	**$80,000**	*Enterprise — typically ELA*
1 PB	**$100,000 (capped)**	*Array-displacement scale*

Multi-year ELAs available for deployments above 500 TB. Typical structure: 3-year commitment, paid migration services ($50K–$250K depending on scope), reference rights, committed growth bands.

Perpetual license available for procurement environments that prefer CapEx accounting. Priced at 3× annual list plus 18% annual maintenance. Deployments above 1 PB remain at $100,000/year.

Next steps

1. **Request the briefs** for your technical and security teams: Architecture Deep-Dive, Security & Compliance Brief, Engineering Maturity Brief. Available under NDA.
2. **Schedule a 30-minute call** to discuss your storage baseline, refresh timeline, and proof-of-concept scope.
3. **Run the POC** for 2–4 weeks (cloud studio) or 4–8 weeks (enterprise). Measure on your hardware, your access patterns, your real depot.

Calculate your savings · Request an evaluation · Book a technical call

info@p4-cache.com *p4-cache.com*