Translytical Processing via DB-OS Co-designed Buffer: Cross-Engine Isolation and Tunable Update Visibility for HTAP

DONGKWANG KIM, Seoul National University, Republic of Korea
KEONWOOK PARK*, Seoul National University, Republic of Korea
CHEOLMIN CHOI*, Seoul National University, Republic of Korea
HYUNGSOO JUNG†, Seoul National University, Republic of Korea

Single-node HTAP increasingly co-locates a row-store OLTP engine with a vectorized OLAP engine over the same logical database. In practice, however, systems either share a monolithic buffer cache, often incurring contention, cache pollution, and unpredictable interference, or decouple engines via log shipping and materialization pipelines, trading freshness for operational isolation. We present Vista, a DB–OS co-designed buffer virtualization framework that explores a middle ground via segment-granular sharing. Vista organizes buffer memory into huge-page-backed virtual segments: OLTP executes over *active* segments, which are periodically *sealed* into immutable regions and remapped read-only into the OLAP process. This design enforces physical isolation between ongoing OLTP writes and OLAP scans while exposing recent updates with *bounded and tunable* visibility lag determined by the sealing policy. A user-level library exports snapshot metadata at sealing boundaries and bridges the row/page–to–vector mismatch via on-the-fly translation with an opportunistic format cache. We implement Vista in PostgreSQL and integrate it with DuckDB. We quantify isolation–freshness tradeoffs under HTAP and show that Vista reduces the interference and improves query latency, achieving up to 12.8× speedup over state-of-the-art baselines.

CCS Concepts: • **Information systems → DBMS engine architectures**; **Online analytical processing engines**; **Record and buffer management**; *Database transaction processing*.

Additional Key Words and Phrases: DB-OS Co-design, HTAP, Performance Isolation, Data Freshness

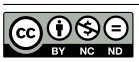
ACM Reference Format:
Dongkwang Kim, Keonwook Park, Cheolmin Choi, and Hyungsoo Jung. 2026. Translytical Processing via DB-OS Co-designed Buffer: Cross-Engine Isolation and Tunable Update Visibility for HTAP. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 246 (June 2026), 27 pages. https://doi.org/10.1145/3802123

1 INTRODUCTION

In applications such as real-time fraud detection, hybrid transactional/analytical processing (HTAP) [31], increasingly framed as translytical processing [33, 40, 45], aims to run analytical queries over continuously changing transactional data with low latency. In practice, however, production deployments still commonly separate OLTP and OLAP engines and synchronize them via log shipping, incremental replication, or ETL pipelines [1, 9, 15, 17, 18, 22, 30, 38]. These designs offer

*These authors contributed equally to this research.
†Corresponding author

Authors' Contact Information: Dongkwang Kim, dongkwang.kim@snu.ac.kr, Seoul National University, Seoul, Republic of Korea; Keonwook Park, keonwook.park@snu.ac.kr, Seoul National University, Seoul, Republic of Korea; Cheolmin Choi, cheolmin.choi@snu.ac.kr, Seoul National University, Seoul, Republic of Korea; Hyungsoo Jung, hyungsoo.jung@snu.ac.kr, Seoul National University, Seoul, Republic of Korea.



ACM 2836-6573/2026/6-ART246
https://doi.org/10.1145/3802123

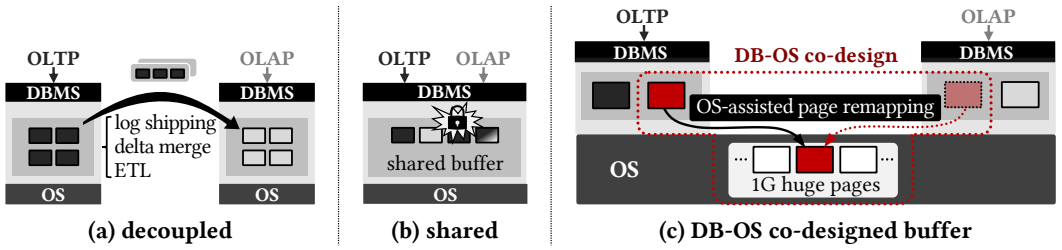


Fig. 1. Architectural comparison for translytical processing: decoupled, shared-buffer, and Vista.

strong operational isolation, but they make *freshness* a first-class cost: analytical results are often computed over delayed snapshots, and data movement/reformatting becomes a persistent overhead.

A natural alternative is to co-locate an OLTP engine and a vectorized OLAP engine on a single server and let the OLAP engine read *close* to the OLTP state. Recent prototypes (e.g., Hermes [26], TwistedTwin [44]) and community efforts around PostgreSQL–DuckDB interoperability [41, 42] underscore both the appeal and the difficulty of this approach. Once two heterogeneous engines share a dataset, the system must confront a three-way tension: (i) *freshness* (how quickly updates can become visible to analytics), (ii) *isolation* (how strongly OLTP and OLAP interfere in CPU, memory, and buffer behavior), and (iii) *format interoperability* (row-oriented OLTP pages vs. columnar OLAP execution). Existing designs typically prioritize two of these goals at the expense of the third. Decoupled pipelines, as shown in Figure 1(a), protect OLTP and OLAP from each other but accept staleness and redundant work; monolithic shared-buffer approaches, as illustrated in Figure 1(b), provide fresher reads but can suffer severe contention, cache pollution, and unpredictable latency under mixed workloads [31, 32, 34].

This paper revisits the buffer cache as the *integration point* for single-node, cross-engine HTAP and asks a more specific question: *What opportunities and tradeoffs arise if we redesign the buffer cache to make freshness and isolation explicit and tunable, rather than implicit and uncontrolled?* Our starting observation is that modern OS virtual memory already provides mechanisms—coarse-grained mapping, protection, and page-table updates—that can enforce strong *physical* isolation while still enabling controlled data sharing. The challenge is that naive, fine-grained remapping at 4 KB granularity is too expensive for high-throughput DBMS execution (e.g., TLB disruption), and freshness control requires carefully defined visibility boundaries between engines.

To explore this design space, we propose Vista (**V**irtual memory segmentation for **I**solation and **S**ynchronization in **T**ransactional and **A**nalytical engines), a DB–OS co-designed framework that virtualizes the buffer cache using *segment-granular* memory management. Vista organizes database buffer memory into 1 GB virtual memory segments backed by huge pages and assigns segments to engines with exclusive access during normal execution. OLTP threads write into *active* segments; when segment buffer utilization reaches a high-water mark, the segment is *sealed* (made immutable) and becomes eligible for (i) asynchronous flushing and (ii) read-only remapping into the OLAP process. By exposing only sealed segments, Vista provides the OLAP engine with a stable in-memory view of recent updates while preserving physical isolation from ongoing OLTP writes acting on a *newly provisioned* segment (Figure 1(c)).

Vista does *not* claim to eliminate all forms of staleness or "solve isolation" in an absolute sense. Instead, Vista makes two design choices explicit. First, **freshness is segment-granular**: updates become visible to OLAP after the segment is sealed and remapped, so the visibility lag is bounded and can be tuned by the sealing policy (e.g., segment size and sealing threshold). Second, **isolation is primarily physical**: by design, OLAP scans do not contend with OLTP operations on active segments, although the system may still experience shared-resource effects (e.g., I/O bandwidth

saturation). Vista therefore frames single-node HTAP as a ***controllable tradeoff space*** rather than a binary "*shared vs. decoupled*" choice, by establishing the segment as a tunable boundary that provides ***spatial isolation for operations*** and ***temporal bounds for visibility***.

Vista further includes a lightweight user-level runtime library that (i) exports OLTP snapshot information at sealing boundaries and (ii) bridges the format mismatch between PostgreSQL pages and DuckDB's columnar vectors through on-the-fly conversion with an opportunistic format cache. This caching avoids bulk materialization pipelines but introduces its own tradeoffs: cached columnar regions can be invalidated by ongoing updates, and tighter freshness can increase invalidation frequency and recomputation. A key goal of our evaluation is to quantify these costs and illustrate when segment-based sharing is advantageous.

We implement Vista as a user-space library alongside a small Linux (6.12) module and integrate it with PostgreSQL and DuckDB. Using CH-benCHmark [7], HyBench [46], and HATtrick [25] under controlled CPU and memory budgets, we evaluate Vista across the isolation–freshness spectrum and quantify how on-the-fly format conversion and caching interact with update intensity. Across our configurations, Vista substantially reduces OLTP–OLAP interference and improves end-to-end analytical latency (up to 12.8× over the compared baselines in our setup), while making the remaining sources of staleness and contention explicit.

In summary, this paper makes the following contributions.

- **Segment-granular buffer virtualization for cross-engine HTAP.** We introduce a buffer design based on 1 GB virtual memory segments that enables physically isolated execution domains for OLTP and OLAP while allowing controlled sharing through remapping.
- **A visibility protocol that exposes committed updates at sealing boundaries.** Vista exports snapshot metadata at sealing boundaries and remaps sealed segments read-only into the OLAP process, yielding bounded, tunable visibility lag without log replay.
- **A cross-engine runtime for snapshot-consistent reads and format bridging.** We integrate PostgreSQL snapshot semantics and on-the-fly row-to-column conversion with a format cache, and we analyze how caching interacts with update rates.
- **Implementation and evaluation on modern storage.** We implement Vista on Linux and quantify the isolation–freshness tradeoffs and overheads using PostgreSQL and DuckDB under CH-benCHmark, HyBench, and HATtrick.

2 BACKGROUND AND MOTIVATION

In monolithic DBMS architectures, the shared buffer cache is primarily optimized for OLTP: it maximizes transaction-local reuse, amortizes I/O, and provides a centralized point for page pinning and writeback. Co-locating a vectorized OLAP engine with an OLTP engine changes the role of this cache from an internal optimization to a cross-engine integration boundary. Beyond efficiency, the system must now define how analytical queries observe recently committed updates (*freshness*) while limiting OLTP–OLAP interference in the buffer path (*isolation*), and it must do so despite semantic and layout differences between engines. This section revisits the principles that shaped conventional buffer design and motivates abstractions that make the isolation–freshness tradeoff explicit in cross-engine HTAP settings.

2.1 The Golden Rule of Buffer Management

For decades, database buffer management has followed the *five-minute rule* [12], which advocated caching pages likely to be reused within five minutes—an economic heuristic rooted in magnetic disk costs. Later refinements [2, 10, 11] adapted this principle to newer storage technologies, reinforcing the shared buffer pool as a core reuse mechanism. However, with PCIe 5.0+ NVMe

SSDs and DRAM-class persistent memory, these assumptions are weakening: reloading analytical data from storage is now cheap, while evicting hot OLTP pages remains costly. As HTAP loads grow more heterogeneous and throughput-intensive, shared buffers in single-server systems fail to reconcile conflicting access patterns—leading to contention, eviction, and delayed visibility.

2.2 The Isolation–Freshness Tradeoff in Cross-Engine HTAP

To mitigate OLTP–OLAP interference, prior HTAP systems have adopted isolation-oriented mechanisms such as partitioned buffer pools [5, 19] and decoupled execution paths [8, 21, 38]. Some main-memory DBMSs (e.g., Hyper [16] and Anker [36]) explored virtual memory snapshotting techniques for HTAP workloads, which use `fork()` or *rewiring* [35]; however, these systems can only handle in-memory datasets by design. The prior approaches improve performance predictability by reducing direct contention between engines, but they also reshape the visibility boundary between transactional updates and analytical reads. In particular, they expose a recurring tension between *isolation* (how strongly OLTP and OLAP are separated in the buffer path) and *freshness* (how quickly committed updates become visible to analytics), which raises questions for any attempt at unified buffer management. We highlight two challenges that repeatedly arise across this design space:

- **Cross-engine visibility semantics.** When OLTP and OLAP are separated (e.g., via distinct processes or buffer regions), OLAP queries still require a well-defined *visibility cut* over the OLTP state, including which recent updates are visible at query start. Many designs establish this cut via (i) *bulk copying*, (ii) *streaming change deltas*, or (iii) *bespoke cross-engine coordination*. Each approach trades off *freshness* against *isolation* and often yields a *visibility lag* that is largely dictated by the mechanism, making it difficult to *tune* to workload needs.
- **Propagation costs for fresher updates.** To make recent updates visible to the analytical path, decoupled designs typically propagate changes via log shipping/replay, incremental replication, or materialization. Although these mechanisms help preserve isolation between engines, they add work in the data path (I/O, CPU, and often format conversion) and impose a non-zero visibility lag depending on the chosen propagation policy.

Recent work on heterogeneous HTAP designs [3, 20, 26, 41, 42, 44] underscores that analytical *freshness* and transactional *isolation* are often coupled by design. As storage bandwidth increases and scan execution gets faster, stale snapshots become less appealing; yet naïvely sharing a buffer cache can expose OLTP to cache pollution, latch contention, and unpredictable interference from long-running scans. Existing systems, therefore, span a spectrum: decoupled designs strengthen isolation but typically incur visibility lag and redundant work (e.g., log replay and reformatting), whereas shared-buffer designs improve freshness but often amplify interference and complicate cross-engine semantics. This paper revisits the buffer cache as an integration point where this tradeoff can be made *explicit and tunable*.

2.3 Time to Rethink the Buffer Manager

To navigate the isolation–freshness tension in single-server, cross-engine HTAP, we introduce Vista, a buffer-management framework centered on *virtual memory segmentation*. Vista leverages OS-supported 1 GB huge pages as coarse-grained, remappable units (segments, `VMSegs`) and uses segment ownership to separate OLTP writes from OLAP scans during normal execution. Rather than claiming that isolation or freshness is *fully solved*, Vista treats both as first-class design dimensions and exposes concrete knobs—most importantly, *when segments are sealed and remapped*—that determine the visibility lag seen by OLAP queries.

2.3.1 Design overview. Vista realizes the isolation–freshness spectrum through three explicit mechanisms.

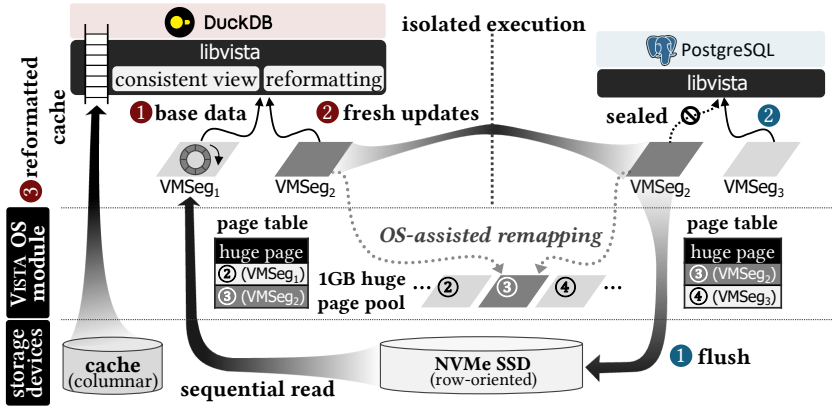


Fig. 2. VISTA design overview.

1. *Seal-to-Bound*: *consistency without tight coordination.* VISTA groups recent OLTP updates into a writable segment. OLTP updates are then *sealed* and exposed to OLAP only at *sealing boundaries* as they are *flushed* to storage. This replaces fine-grained, scan-time coordination between OLTP transactions and OLAP queries with a coarse-grained visibility rule: OLAP reads from sealed (immutable) segments, while OLTP continues updating on active segments. The resulting visibility lag is *segment-granular* and can be tuned via the sealing policy, making the isolation–freshness tradeoff explicit.

2. *Remap-to-Share*: *zero-copy access to sealed state.* After sealing, the VISTA OS module maps the *sealed* OLTP `VMSeg` into the OLAP address space, enabling zero-copy access to committed in-memory state without bulk replication or log replay. Nevertheless, OLAP queries may still pay per-access overheads—most notably, tuple filtering under the exported visibility snapshot information and row-to-column conversion for vectorized execution.

3. *Scan-then-Patch*: *storage-first scans with a sealed update window.* In cross-engine HTAP, where OLAP reads from the same storage that OLTP updates and flushes, OLAP workers stream the bulk of their input from fast storage via sequential I/O and consult a remapped, sealed `VMSeg` to incorporate the most recent updates while dirty pages in the `VMSeg` are being flushed. Because the sealed `VMSeg` serves as an in-memory update window, OLAP does not read disk pages that are being persisted, thereby avoiding the risk of observing partially persisted page states.

Overall, VISTA repositions the buffer manager from a reuse-optimized page cache into a cross-engine substrate where *visibility* and *ownership* are managed at segment granularity. Through DB–OS co-design, VISTA repurposes OS primitives (page-table remapping, huge pages, and memory protection) to (i) enforce *physical* separation between OLTP writes and OLAP scans on active data, (ii) expose committed state through *sealed, read-only* segments with bounded visibility lag, and (iii) keep the control plane lightweight enough to be practical in a high-performance DBMS.

2.3.2 Translytical data access paths in VISTA. VISTA structures translytical execution into two distinct access paths (Figure 2) and realizes the design principles. On the OLTP side (e.g., PostgreSQL), VISTA applies *generational* buffer management to OLTP-owned `VMSegs`, implementing '***Seal-to-Bound***': when a sealing criterion is met, the current `VMSeg` is sealed and becomes eligible for asynchronous flushing (Figure 2-①), while a new `VMSeg` is provisioned as an active segment for continued OLTP updates (Figure 2-②), sealing the current generation and beginning a new one.

On the OLAP side (e.g., DuckDB), analytical queries use private ring buffers allocated in an OLAP-owned `VMSeg` and exploit sequential access patterns. Following '***Scan-then-Patch***', most input is streamed directly from fast storage—often bypassing the OS page cache (Figure 2-①). To

incorporate recent updates, OLAP scan workers consult sealed state via '***Remap-to-Share***': they access the sealed VMSeg remapped read-only into the OLAP address space, which serves as an in-memory update window for the immediately previous generation (Figure 2-❷). When accessing remapped segments, the Vista runtime library (libvista) applies Multi-Version Concurrency Control (MVCC) visibility filtering, performs on-the-fly row-to-column conversion for vectorized execution, and optionally caches the resulting intermediate results for reuse (Figure 2-❸).

3 Vista ARCHITECTURE

This section details the architecture and implementation of Vista, focusing on how it enables cross-engine data access with policy-controlled visibility lag. We first describe the structure and lifecycle of VMSegs (§3.1). We then trace how OLTP updates become visible to OLAP with OS assistance, starting with OLTP generational buffer management with tunable sealing policy (§3.2), then OS-level mechanisms for remapping (§3.3), and finally OLAP hybrid scan over storage and in-memory update window (§3.4).

3.1 Segment Memory Management

3.1.1 Segment memory organization. A VMSeg is a 1 GB contiguous virtual memory region backed by a single 1 GB huge page, serving as the unit of cross-engine sharing. While all VMSegs share the same underlying physical properties, Vista configures them into three distinct functional roles, each with a specialized internal layout and management policy:

- **OLTP-managed segments:** Vista allocates multiple VMSegs to serve as the primary transactional buffer pool. Internally, this region embeds its own page descriptors, buffer frames, hash table, and any other necessary data structures (Figure 3 left). This *self-contained* layout allows the entire region to be updated by OLTP workers and subsequently remapped into the OLAP engine as a stand-alone, read-only unit without external dependencies.
- **OLAP-only segment:** For analytical processing, Vista allocates a dedicated VMSeg to contain simple, high-throughput ring buffers. This segment is strictly private to the OLAP engine and is optimized for buffering sequential reads from storage.
- **System metadata segment:** Internal DBMS metadata (e.g., system catalogs, indexes) requires random access but not cross-engine visibility. For these structures, Vista allocates a separate VMSeg that hosts a traditional shared buffer pool managed via standard LRU-based eviction.

To coordinate these physically isolated segments, Vista establishes a lightweight control plane: the **shared metadata registry** (Figure 3 bottom right). Residing in a small shared memory region (e.g., 16MB), this registry serves as the authoritative source for cross-engine synchronization. It houses critical control structures including state flags for segment lifecycle management, `readview` (MVCC snapshot data) for consistent visibility, global generation number (`global_generation`), and address offsets required to resolve pointers within remapped segments.

3.1.2 Segment memory lifecycle. For the segment-granular memory management model, Vista imposes a four-stage lifecycle for every OLTP-managed VMSeg: *ready, active, sealed,* and *flushed* (Figure 3 top right). ***(i) ready.*** At startup, the OLTP engine acquires a pool of VMSegs via the `vista_mmap()` system call and initializes them. These segments are mapped into the OLTP engine's address space but remain logically empty, transitioning to *active* when the engine designates the segment as the active buffer pool. ***(ii) active.*** The OLTP engine, with exclusive ownership of the VMSeg in this phase, utilizes it as the operational buffer pool. When a configurable sealing criterion is met—either reaching a utilization threshold (*capacity-based*) or a time limit (*time-based*)—the segment transitions to *sealed.* This policy explicitly defines the freshness window: lower thresholds yield fresher OLAP views at the cost of more frequent flushings. ***(iii) sealed.***

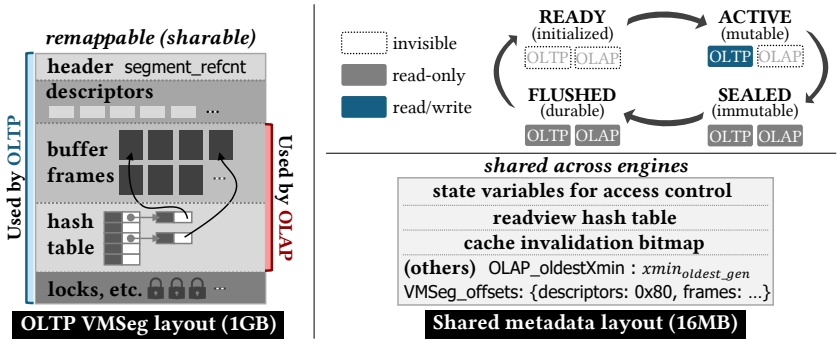


Fig. 3. OLTP-managed VMSeg layout (left), VMSeg lifecycle (top right) and metadata layout (bottom right).

Vista enforces immutability on the *sealed* segment, protecting it from further modification. This enables its safe remapping into the OLAP address space and the asynchronous flushing of the updates. Consequently, the segment becomes visible to both engines with read-only permissions. ***(iv) flushed.*** Once the segment is fully synced to storage, it transitions to the *flushed* state, after which the OLAP engine can explicitly unmap it. Following this, a `libvista` background worker initiates a cleanup to return it to the *ready* pool for reuse. The underlying segments are released back to the OS kernel only after the DBMS engines terminate and call `vista_munmap()`.

3.2 OLTP: Generational Buffer Management

In Vista, the OLTP engine uses the VMSeg abstraction to convert a continuous stream of transactional updates into discrete, immutable units called *generations*. A generation comprises the updates accumulated in an *active* VMSeg before it is sealed. Vista enforces generation boundaries via *sealing*, which transitions an *active* VMSeg into an immutable state and directs subsequent updates to a newly provisioned active segment.

3.2.1 OLTP execution over generational segments. In Vista, the OLTP engine executes transactions over pages residing in an *active* VMSeg. Unlike traditional buffer pools that rely on LRU-style eviction, Vista adopts a *no-eviction* policy and allocates buffer pages sequentially within the *active* segment on cold-page accesses. As a result, the active generational buffer pool has a hard capacity limit: it cannot reclaim individual pages by syncing and recycling them in place. Sealing must therefore be triggered before the segment is exhausted. In Vista, an OLTP worker monitors segment occupancy during buffer allocation and, upon crossing a threshold, signals the `libvista` background worker to initiate sealing; alternatively, the background worker can trigger sealing on a timeout. Once sealing begins, Vista increases `global_generation` and designates a *ready* VMSeg as the next active generational buffer pool, from which OLTP allocates subsequent buffer pages.

To transition a segment from *active* to *sealed* safely in the presence of concurrent transactions, Vista tracks pinned pages using a per-segment reference count. After sealing is triggered, no new buffer pages are allocated from the *sealed* segment, so the reference count eventually drops to zero. The background worker then finalizes the seal, invokes remapping, and asynchronously flushes all dirty pages. A critical challenge arises when a transaction needs to update a page in a *sealed* segment before it has been fully flushed. Without intervention, the writer would either block until the flush completes or fall back to fetching a stale copy from disk. Vista handles this case with a ***Copy-on-Seal*** policy (Figure 4 right): the transaction copies the target page from the *sealed* segment into the current *active* segment and then modifies the copied page. Although this incurs `memcpy()`, it can benefit OLTP with high temporal locality by migrating hot pages across generations, keeping frequently accessed data in memory, and avoiding blocking during the flush.

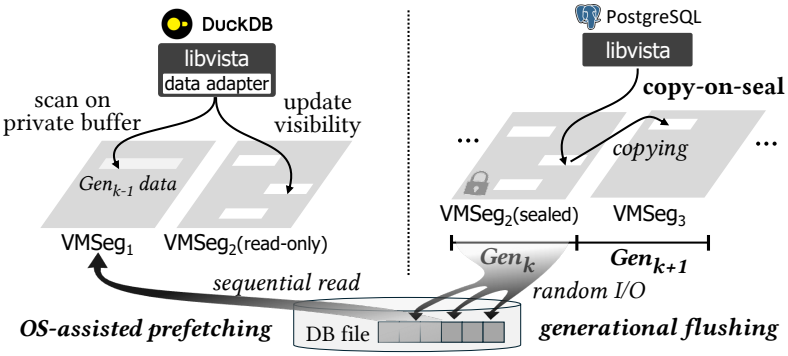


Fig. 4. OLAP and OLTP access paths in Vista.

3.2.2 Freshness knob. A fundamental tradeoff in Vista's segment-granular model is that OLTP updates become visible to the analytical engine only at sealing boundaries. As a result, the rate at which the active segment fills (i.e., buffer allocation and update intensity) affects how quickly new updates are exposed (i.e., data freshness). To mitigate staleness under low-write loads, Vista exposes the sealing policy as a **tunable freshness knob**. In particular, Vista supports two triggers:

- **Capacity-based sealing (*default*).** Vista seals a segment when its buffer-usage ratio ($\frac{\#\text{allocated buffers in VMSeg}}{\#\text{total buffers in VMSeg}}$) exceeds a configurable high-water mark (e.g., 90%). This favors memory utilization and I/O efficiency, but at low write intensity, it may seal less frequently, resulting in lower freshness.
- **Time-based sealing (*timeout*).** To bound the update visibility lag under low update rates, Vista also supports a timeout trigger in the background worker. Even if the buffer-usage ratio remains low, the worker seals the *active* segment after a configurable interval (e.g., 1 s) elapses, keeping visibility lag under control.

By allowing database administrators to configure these triggers, Vista lets deployments choose where to operate on the isolation–freshness spectrum; favoring higher OLTP throughput with less frequent sealing, or tighter and more predictable OLAP freshness with more frequent sealing.

3.3 OS-assisted Segment Remapping

Vista exploits the architectural property that a single huge page can represent a contiguous memory region (1 GB), allowing the entire `VMSeg` to be remapped via a single operation. It modifies a single pud (Page Upper Directory) entry—each of which maps a contiguous 1 GB region in `x86_64` systems with 5-level page tables—to expose the past generation's updates to the OLAP engine. This coarse-grained approach eliminates the need for fine-grained page table updates, ensuring minimal TLB disruption. While our prototype targets `x86_64` platforms, the entire remapping protocol is designed to be architecture-neutral, relying solely on Linux virtual memory and huge page support. Figure 5 illustrates the overall sequence of OS-assisted safe segment remapping.

3.3.1 Vista runtime support for segment remapping. Vista reserves a configurable number of 1 GB huge pages at Linux kernel boot time, which serve as the physical memory backing for all `VMSegs`. To exploit NUMA locality, huge pages can be reserved per NUMA node, enabling dedicated memory regions for OLTP and OLAP engines. When an engine requests new `VMSegs` (see §3.1.2), these huge pages are mapped into its virtual address space on demand. Once mapped, the OLTP engine directs all transactional writes to the active segment. The remapping process begins when sealing is initiated, either by an OLTP worker upon crossing a utilization threshold (Figure 5-❶) or by the `libvista` background worker on a timeout. Once the segment reaches a quiescent state, the

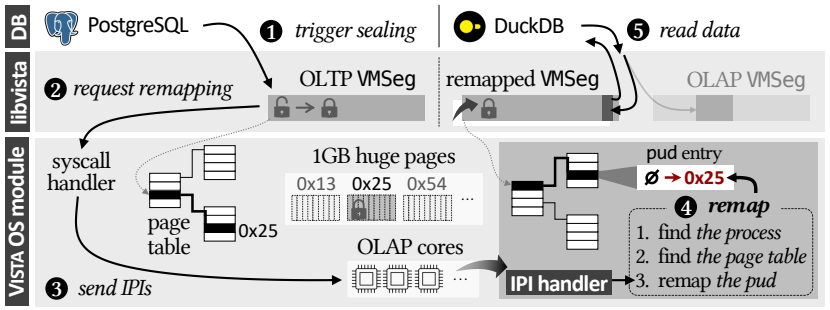


Fig. 5. OS-assisted safe segment remapping in VISTA.

`libvista` worker invokes a dedicated system call, `vista_remap()`, to initiate OS-assisted segment remapping (Figure 5-❷). The OS module then modifies a single Page Upper Directory (PUD) entry per OLAP process, mapping the physical frame of the 1 GB segment into the OLAP address space. This coarse-grained remapping avoids the overhead of per-page updates, enabling safe data sharing across isolated engines.

3.3.2 OS support for safe, controlled remapping. To propagate the new mapping across CPU cores, the OS module issues an *inter-processor interrupt* (IPI) from the OLTP core (Figure 5-❸), a hardware mechanism that allows one processor to signal another in a multiprocessor system. VISTA installs a custom IPI handler to notify relevant OLAP threads of remapping events. Upon receiving an IPI, the handler locates the target OLAP process or thread and updates its page table by inserting a new PUD entry that maps the sealed 1 GB segment into the OLAP virtual address space with read-only permissions (Figure 5-❹). Once this operation completes, the OLTP-generated segment becomes immediately visible to OLAP workers. To maintain consistency without polling, the IPI handler also updates the `remap` flag in the shared metadata registry, effectively notifying OLAP workers that fresh data is available (Figure 5-❺). This enables OLAP workers to react promptly without synchronization. Since remapping occurs at 1 GB granularity using huge pages, the associated TLB shootdown costs are minimal and amortized across large memory regions.

3.3.3 Access control and revocation. OLAP workers must not only be able to access the remapped segment during scanning but also promptly cease reading updates to ensure timely reclamation. VISTA coordinates this through a cooperative access protocol that operates at the granularity of individual page accesses.

First, OLAP workers check the `remap` flag and the segment state on every `get_page()` call (see Figure 6). This allows the OLAP engine to discover newly remapped updates and switch to them mid-scan. When the flag is set, the *first* OLAP worker attaches to the segment and sets the `remap_attached` flag so that subsequent workers can proceed without repeating the attachment work. After the background flush completes, the segment state transitions to *flushed*. Active OLAP readers observe this transition on their next `get_page()` call and stop accessing the remapped segment. This design avoids starvation: readers exit after completing their current page access (typically 8 KB). Finally, the *last* OLAP worker to leave a flushed segment detaches by invoking the `vista_unmap()` custom system call to release the virtual mapping and clear the `remap_attached` flag. This signals to the background worker that it is safe to clean up and recycle the segment.

3.4 OLAP: Hybrid Data Access

VISTA provides a *hybrid data access path*: a lightweight routing mechanism that directs each OLAP page request to the appropriate physical location, depending on whether the page is served from

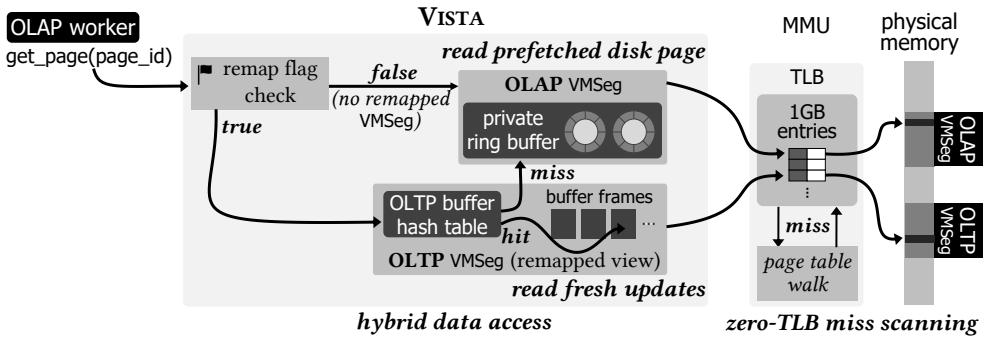


Fig. 6. OLAP data access in local/remapped VMSegs.

persistent storage or from a remapped sealed segment. By exposing a single `get_page()` interface, the runtime hides the underlying placement and remapping details from the OLAP engine. The `remap` flag check (§3.3.3) steers each request onto the correct path. As a result, analytical queries incorporate the most recent *available* updates at page granularity, while avoiding reads from pages that are concurrently being persisted during asynchronous flushes.

3.4.1 On-disk data: sequential scan on local VMSeg. For on-disk data from prior generations, VISTA uses high-throughput private ring buffers optimized for sequential access (Figure 6, top). This is the default path when no remapped VMSeg is available. It exploits the underlying *heap-file organization* (or, more generally, any row-store layout that supports sequential heap scans) [13, 23, 28, 29], in which the physical data blocks solely represent data tuples.

Each OLAP worker allocates a private ring buffer inside the OLAP-only VMSeg. Since analytical queries exhibit predictable, scan-oriented access patterns, the runtime triggers asynchronous I/O prefetches in the background, populating read-only scan buffers without blocking the main query execution path. This prefetching is driven by logical page IDs and performed entirely in user space, enabling low-latency data access without OS intervention. As the scan window advances, `libvista` recycles buffer entries in circular order, ensuring bounded memory usage while preserving fast lookup and page reuse for prefetching.

3.4.2 In-memory updates: hash table lookup on remapped VMSeg. Once an OLTP VMSeg is remapped, the most recent (*fresh*) updates become visible to the OLAP engine as a segment-granular batch. VISTA then transparently redirects OLAP reads for pages updated in that generation to the remapped segment. This relies on the invariant that any page modified during the sealed generation resides in the corresponding sealed VMSeg (and is therefore available through the remapped view).

When the `remap` flag is set, an OLAP worker first probes the remapped segment's buffer hash table using the requested page identifier. On a hit, the worker reads the page directly from the remapped OLTP VMSeg. On a miss, the worker proceeds with the prefetched data provided that the prefetching operation did not overlap with the flushing period; this validation prevents the reading of torn writes. Otherwise, a worker might retrieve an inconsistent, partially updated page.

Crucially, accessing a remapped VMSeg requires *address translation* to bridge the distinct virtual address spaces of the OLTP and OLAP processes. Because the segment may be mapped at a different virtual base in the OLAP process, VISTA performs *initial offset reconstruction* and *dynamic pointer resolution*. Upon initial attachment (see §3.3.3), the OLAP worker reads offset metadata from the shared registry to reconstruct the virtual addresses of key in-segment structures (e.g., buffer frames and the buffer-hash-table control block). This reconstruction is performed once per remapped segment. Subsequently, during hash-table traversals, internal pointers (e.g., bucket-chain links) are

resolved on the fly by converting an OLTP-relative pointer into an OLAP address: Vista subtracts the OLTP segment base and adds the OLAP segment base.

3.4.3 Zero-TLB miss scanning and data sharing. The 1 GB huge pages backing `VMSegs` avoid fine-grained page faults and reduce TLB pressure during large sequential scans. Moreover, remapping operates at the granularity of a huge page: in our prototype, the Vista OS module updates a single pud entry to install the new mapping. As a result, the required TLB shootdown invalidates only a single entry rather than a quarter million 4 KB entries, and the remapping overhead is amortized over the entire 1 GB region.

4 DATABASE INTEGRATION

Vista enables seamless cross-engine operation by providing a minimally intrusive runtime library that integrates at the buffer management layer without modifying core query processing in either engine. Although segment-granular buffer management addresses the *physical* data-access path, integrating heterogeneous OLTP/OLAP engines requires handling two additional concerns:

- ***Cross-engine snapshot semantics*:** Analytical queries require a stable snapshot of the database (e.g., snapshot isolation). Since the multiversioned state resides in the OLTP engine, Vista must export snapshot metadata (e.g., `readviews`) so that the OLAP engine can apply snapshot-consistent visibility rules over remapped segments and storage-resident data for `REPEATABLE READ`.
- ***Format interoperability*:** To support vectorized execution, Vista must bridge the layout mismatch between OLTP row stores and OLAP columnar vectors. Bulk conversion resembles ETL-style materialization and can introduce high latency, while purely on-the-fly conversion can repeat work across queries. Moreover, ongoing OLTP updates change visibility and can invalidate previously converted results. Vista therefore needs a conversion strategy that amortizes materialization through reuse while efficiently handling partial invalidation.

This section describes how the Vista runtime provides cross-engine snapshot consistency (§4.1) and implements on-demand format conversion with an opportunistic format cache (§4.2). We then present integration details for PostgreSQL and DuckDB (§4.3).

4.1 Cross-Engine Snapshot Isolation

Snapshot isolation ensures that a query observes a consistent view of the database as of a chosen snapshot time. In MVCC systems, this is commonly implemented via a `readview` (snapshot) that summarizes concurrent transaction IDs (e.g., active `XIDs`). The query uses this snapshot to filter tuple versions, excluding versions created by transactions that were active at the snapshot time or that started after it. Vista adopts *generational readviews*: instead of creating a `readview` at arbitrary query start times, Vista aligns snapshot creation with `VMSeg` generation boundaries. This approach builds on standard MVCC semantics and applies to DBMSs that implement snapshot-based visibility.

Snapshot export at sealing boundaries. Vista captures a snapshot whenever a generation boundary is crossed, i.e., when a `VMSeg` is sealed and remapped. In PostgreSQL, transactional visibility is defined by a `SnapshotData` structure which encodes the set of active transaction IDs at a particular snapshot timestamp. Vista leverages internal PostgreSQL APIs to export this `readview` to a file. To make this snapshot discoverable, the runtime maintains a compact hash table in the shared metadata registry using the sealed segment's generation number as the key and the `readview` file name as the value (see Figure 3). This indirection enables each OLAP worker to retrieve the exact snapshot for the generation to which it belongs.

Snapshot reconstruction during scan. Prior to the columnar transformation of PostgreSQL's heap data, the OLAP-side runtime enforces snapshot isolation through a coordinated visibility

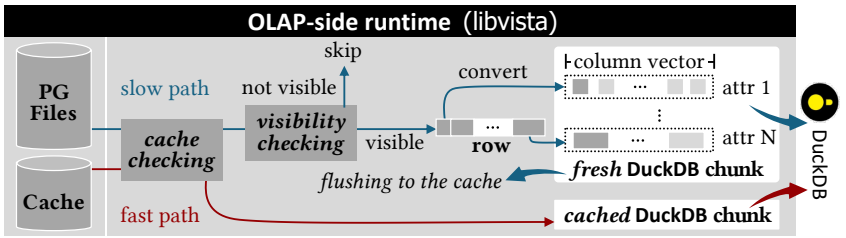


Fig. 7. Data reformatting and caching in `libvista`.

check. The process begins by assigning each OLAP query a target generation—specifically, the value of `global_generation` − 1 at the query start time. Using this as a lookup key, the runtime retrieves the corresponding `readview` file from the lookup table maintained in the shared metadata registry. This `readview` contains the visibility information captured at the time the associated segment was sealed by the OLTP engine, and thus serves as the authoritative visibility snapshot for that generation. The Vista runtime then reconstructs PostgreSQL's MVCC semantics—specifically the logic of `HeapTupleSatisfiesMVCC()`—by applying this `readview` in conjunction with auxiliary structures such as the commit log, `MultiXacts`, and `Subtrans`.

4.2 Format Interoperability

By applying the OLTP engine's MVCC rules using the generational `readview` to enforce precise snapshot semantics, Vista ensures that only visible tuples are retained for the OLAP query's snapshot read. For these visible tuples, the runtime dynamically converts the row-oriented layout used by OLTP into the columnar format expected by the OLAP engine. To amortize the cost of repeated transformations, Vista opportunistically caches the reformatted columnar output and invalidates the cache for updated data at generation boundaries. This cache not only accelerates future analytical queries over the same data but also facilitates the efficient reuse of visibility filtering results, significantly reducing query response time under repeated access patterns. Although the cache can theoretically grow to the size of the database, it operates strictly as an opportunistic accelerator; it can be disabled without affecting correctness. We demonstrate that even under heavy OLTP update workloads—where frequent cache invalidations occur—Vista maintains robust analytical performance, as detailed in our evaluation (§5).

Data format conversion. Once a tuple passes the visibility check, the OLAP-side runtime converts PostgreSQL's native row layout into DuckDB's columnar format. To enable this reformatting, Vista leverages format conversion routines from DuckDB's PostgreSQL extension [41]. As illustrated in Figure 7, the runtime deserializes PostgreSQL's slotted page layout on-the-fly and materializes the visible records as columnar vectors within DuckDB's native *chunk collection* abstraction. Each thread in DuckDB is assigned a disjoint region of the table, referred to as a *block group* (e.g., 128 MB), and independently performs both file I/O and format conversion for that region. Upon completing its assigned block group, the thread dynamically acquires the next unprocessed region in a work-stealing manner. This parallelized, chunk-at-a-time strategy enables efficient columnar ingestion.

Format caching and invalidation. To avoid redundant conversions, Vista employs a *persistent format cache* that stores reformatted columnar data on a dedicated storage device. However, ensuring consistency requires that cached columnar data be invalidated whenever the OLTP engine updates the underlying row-oriented data. To maximize reuse while minimizing invalidation overhead, Vista tracks validity at a fine granularity: each block group is subdivided into subgroups (e.g., 16 MB). This prevents minor OLTP updates from invalidating entire block groups.

To facilitate lock-free coordination of cache states, the OLTP-side runtime maintains a double-buffered *cache invalidation bitmap* for each table. Each bit represents the synchronization status of a subgroup; `0` indicates the cache is consistent and safe to use, while `1` indicates it is invalid due to recent updates. The scheme utilizes two distinct bitmaps: an *active* bitmap (read and updated by OLAP workers) and a *shadow* bitmap (written exclusively by the `libvista` background worker). When an OLAP worker reads the active bitmap and encounters an invalid (1) subgroup, it converts the fresh data, caches the outputs, and then resets the bit to `0`. During the flushing of dirty pages at generation boundaries, the background worker clones the active bitmap into the shadow copy, sets the bits for newly dirtied subgroups to 1, and performs an *atomic swap* of the two bitmaps to expose the invalidations to new OLAP queries. Although a benign race may occur—where a bit cleared in the old active bitmap is overwritten by the shadow copy during the swap—this results only in a harmless false positive. The cache remains valid and, although future queries may perform a redundant conversion, stale reads are strictly prevented.

Opportunistic caching. Vista adopts an opportunistic caching policy; instead of mandating universal caching for every converted subgroup, it leverages deterministic opportunities to persist data while reverting to base-data reformatting when state is indeterminate. When an OLAP thread is assigned a block group, it consults the invalidation bitmap prior to *any* data access. This *early validation* prevents I/O stalls by allowing conversion and computation to overlap with disk access. Valid subgroups are loaded from the format cache, whereas invalid or stale ones are fetched from base storage and reformatted on-the-fly. Before caching the newly reformatted outputs, Vista enforces two safety conditions: *(i) Generation consistency*: the active bitmap must not have undergone a swap between the thread's initial read and its attempt to clear the bit. *(ii) Data stability*: the subgroup contains no uncommitted data and no in-flight tuple versions (i.e., every tuple's MVCC state is deterministic for all future snapshots). If both conditions are met, all reformatted columnar vectors are serialized into a single binary segment and batch-written to the cache file alongside per-column metadata to enable future *column-selective reloads*.

4.3 Implementation Details

We implemented Vista over the popular open-source DBMS engines, PostgreSQL and DuckDB.

PostgreSQL integration. The primary role of the OLTP-side Vista runtime is to capture all transactional modifications into the OLTP-managed `VMSegs`, which can be remapped to OLAP engines once sealed. To achieve this in PostgreSQL, Vista initializes a set of distinct buffer pools, each backed by a single `VMSeg`, instead of PostgreSQL's default single buffer pool. The runtime then intercepts buffer allocation requests using a Vista wrapper that differentiates between metadata pages (e.g., catalogs, indexes) and data pages: metadata continues to be allocated from PostgreSQL's legacy shared buffer pool (located in the system metadata segment), while data pages are drawn from the currently active OLTP segment. Crucially, this redirection is entirely transparent to PostgreSQL's execution and storage subsystems; by operating at the buffer memory level, Vista preserves the semantics of PostgreSQL's core mechanisms, including its concurrency control (MVCC), write-ahead logging (WAL), and recovery.

We implemented the `libvista` background worker as a PostgreSQL process. Although Vista's shared-metadata coordination could be driven by an external daemon, running the worker inside PostgreSQL simplifies lifecycle management (startup, shutdown, and failure handling) and reduces deployment complexity. When a segment is sealed, the worker initiates an asynchronous flush to persist the segment's contents. This writeback can be scattered across files and offsets, but modern PCIe 5.0 SSDs can sustain millions of IOPS, making the overhead negligible in practice. To reduce syscall and synchronization overhead, Vista leverages `io_uring` to issue large batches of asynchronous writes with low per-request CPU cost; the worker collects dirty pages from the

sealed segment, groups and orders them by their target data file, and then issues all writes for a file before invoking a single `fsync()` for that file, amortizing synchronization. Using this collected set of dirty pages, the worker updates the shadow bitmap for the format cache and swaps the bitmaps. To ensure cross-engine snapshot isolation, the worker then flushes essential MVCC metadata, including the commit log, `MultiXacts`, and `Subtrans`. This generational flush strategy resembles traditional checkpointing, but avoids its blocking behavior: OLTP threads immediately switch to a newly provisioned active segment, and updates to pages residing in the sealed segment proceed via *Copy-on-Seal* without blocking on writeback.

DuckDB integration. We integrate Vista with DuckDB via its `TableFunction` interface, enabling direct access to PostgreSQL-resident data without requiring modifications to DuckDB's core execution engine. During conversion, the Vista runtime iterates over each PostgreSQL tuple, transforming its attributes into DuckDB's internal columnar representation. This transformation is governed by a type-mapping layer that reconciles differences between PostgreSQL and DuckDB data types, leveraging the existing logic from the open-source `pg_duckdb` project [42]. To execute this conversion accurately, DuckDB must possess both the logical schema and the physical layout metadata of the corresponding PostgreSQL tables. To this end, Vista establishes a control-plane connection to the PostgreSQL backend during initialization, extracts the required catalog metadata (e.g., table definitions, column types, storage offsets), and installs it into DuckDB's internal catalog. This design builds on DuckDB's PostgreSQL extension framework, providing a modular integration point for hybrid access while avoiding invasive code changes. Finally, to reduce interference from high-bandwidth analytical scans, Vista places OLAP-only segments on a NUMA node distinct from PostgreSQL's, thereby isolating OLAP memory traffic from OLTP-critical memory channels. Although OLAP threads may temporarily access OLTP-managed segments during the remapping window, the resulting interference is limited by two factors. First, the remapping period is brief, typically on the order of sub-second latency. Second, the vast majority of OLAP scan traffic is served from the OLAP engine's private ring buffer. For realistic database sizes, a 1 GB `VMSeg` represents only a small fraction of the entire dataset, further bounding the scope of cross-engine interference.

5 EVALUATION

We evaluate Vista to assess its ability to deliver *performance isolation* and *tunable update visibility* across transactional and analytical engines. Our experiments focus on enabling real-time analytics over freshly updated transactional data while maintaining robust performance under varying workload conditions within a single-node *translytical platform*. Our evaluation is structured around the following key questions, designed to assess the end-to-end effectiveness of Vista in enabling translytical processing:

Q1. *How does the tunable freshness knob trade off data freshness against performance isolation?* (§5.2)
We quantify how sealing-timeout settings shift the isolation–freshness tradeoff by measuring visibility lag and HTAP performance isolation across diverse workload patterns.

Q2. *Does Vista preserve OLAP performance under varying OLTP loads?* (§5.3)
We examine whether Vista effectively mitigates resource contention and sustains low OLAP query latency with varying OLTP workloads.

Q3. *How does Vista maintain performance isolation during dynamic HTAP transitions?* (§5.4)
We evaluate the runtime behavior of OLTP and OLAP engines under shifting workloads, analyzing performance trajectories and identifying points of interference or stability.

Table 1. Database and resource configurations.

Baseline Systems		Resources		PostgreSQL Conf.				DuckDB Conf.	
		CPU	Mem.	shared_buffers	work_mem	max_parallel_workers	max_parallel_workers_per_gather	Mem.	Threads
Vanilla PG		32	64 GB	24 GB	256 MB	16	16	-	-
PG (pg_duckdb)		32	64 GB	16 GB	32 MB	8	8	40 GB	16/32
PG+D		32	64 GB	16 GB	32 MB	8	8	40 GB	16/32
Vista	**PG**	32 †	64 GB	3 GB (3 VMSegs)	32 MB	8 (default)	2 (default)	-	-
	DuckDB			1 GB (1 VMSeg)	-	-	-	40 GB	16/32

† PostgreSQL is pinned to the 16 cores of a single NUMA node to avoid cross-node interference, while DuckDB is allowed to utilize all 32 cores to maximize parallelism.

Q4. *What is the runtime overhead of format conversion and visibility enforcement?* (§5.5)
We break down the cost of in-memory format translation and MVCC-based filtering during analytical scans.

Q5. *What is the performance benefit of format caching?* (§5.3)
We assess the effectiveness of Vista's columnar format cache in reducing redundant conversion and I/O volume.

5.1 Experimental Setup

Server configurations. All experiments are conducted on a high-performance server equipped with two 128-core AMD EPYC 9754 processors (2.25 GHz, 256 MB L3 cache), 1.5 TB DDR5 memory, and two 8 TB of PCIe 5.0 NVMe SSD storage devices, each offering 14 GB/s for sequential reads. The SSD subsystem supports a total of 28 GB/s aggregate sequential read bandwidth across two physical devices. The system runs Ubuntu 22.04 LTS with Linux kernel 6.12, and `hugetlbfs` is configured to support 1 GB huge pages.

Computing resource constraints. To reflect realistic single-server deployment scenarios—such as cloud VMs (e.g., Azure standard F32s v2 [24] (or AWS c4.8xlarge [4]) instance featuring 32 (or 36) vCPUs and 64 (or 60) GB memory for compute-intensive workloads) or containerized HTAP setups—we enforce strict resource isolation using Linux `cgroups`. Each experimental run is confined to a `cgroup` with a hard cap of **64 GB** memory and **32 CPU cores**, preventing any system from exploiting host-level resources (e.g., via aggressive use of the Linux page cache). To further ensure fairness and NUMA locality, we employed an asymmetric resource configuration for Vista as indicated in Table 1. This setup minimizes cross-node memory traffic, where OLTP/OLAP engines operate in physically isolated execution domains.

OLTP and OLAP database engines. We evaluate Vista using two widely adopted open-source engines representative of transactional and analytical workloads: PostgreSQL 15.2 and DuckDB 1.2.2. PostgreSQL serves as the OLTP engine, executing transactional workloads using its native concurrency control and write-ahead logging. DuckDB is used as the OLAP engine, executing vectorized scan queries over both disk-resident and remapped data. Both engines are modified to integrate with the Vista runtime via a lightweight user-space buffer API.

Baseline HTAP systems. As baseline HTAP systems with similar engine composition and design philosophy, we construct four configurations based on PostgreSQL and DuckDB to highlight the performance and isolation properties of Vista:

(1) Vanilla PostgreSQL (PG): The unmodified PostgreSQL engine using its default shared buffer pool and LRU-based eviction. OLTP and OLAP are co-located in a single engine and share the same memory space, leading to contention and interference. This setup provides no OLAP-specific optimizations other than ring buffer strategy for bulk read.

(2) **PostgreSQL + Integrated DuckDB (PG (pg_duckdb)):** A tightly coupled hybrid setup where DuckDB's vectorized OLAP engine is embedded into PostgreSQL via the pg_duckdb extension (v1.0.0, released on 2025/09/04) [42]. All analytical queries are automatically accelerated through internal delegation but still depend on PostgreSQL's buffer manager for data access, limiting physical isolation.
(3) **PostgreSQL + ETL to DuckDB (PG+D):** A loosely coupled setup where DuckDB connects to PostgreSQL as an external client, retrieving updates through COPY operations [41]. This configuration ensures physical separation but incurs ETL overhead and materialization delays, sacrificing real-time visibility.
(4) **PostgreSQL to DuckDB via the Vista framework:** A co-designed system where DuckDB directly accesses PostgreSQL segments being flushed via OS-assisted memory remapping. This configuration ensures zero-copy visibility of committed transactional data, physical isolation between engines, while avoiding shared-buffer contention or ETL-induced delays.

Workload configurations. To emulate translytical workloads, we follow TPC-CH [7] and use three HTAP benchmark suites: CH-benCHmark [6] (driven by HammerDB 4.4 [14]), HyBench [46], and HATtrick [25]. For CH-benCHmark, we initialize TPC-C with 1000 warehouses (WH) to stress cross-engine execution at scale. For HyBench, which models banking-style workloads with 18 transactions and 13 analytical queries, we use a scale factor (SF) of 100 and evaluate its OLTP and OLAP components. For HATtrick, which combines three TPC-C–like transactions with analytical queries from the Star-Schema Benchmark (SSB) [27], we use SF 100.

The OLTP components of CH-benCHmark (TPC-C) and HyBench are configured with either uniform or skewed access. In TPC-C, skew is modeled by binding each transactional client to a single randomly chosen warehouse, whereas the uniform configuration distributes accesses across all warehouses. In HyBench, skew is controlled by sampling target record identifiers (e.g., accounts) from either a uniform distribution or a power-law distribution.

Database configurations. We carefully tuned performance-critical parameters—particularly those governing memory allocation and parallel query execution—for each baseline system to ensure optimal performance, as specified in Table 1. Our tuning strategy prioritized balanced parallelism between data retrieval and query processing, considering that, unlike Vista, all baselines must retrieve data through PostgreSQL's shared buffer pool—either directly or through copying—thereby incurring contention and data movement overheads. For example, we set `max_parallel_workers` and `max_parallel_workers_per_gather` to 8 for PG (pg_duckdb) and PG+D systems to enable parallel data retrieval. Moreover, we disabled `autovacuum` in PostgreSQL, since it introduces unpredictable I/O and CPU loads that create performance jitter, obscuring the core architectural differences we aim to measure.

Vista allocates a total of 4 GB of memory for data buffers, comprising three `VMSegs` dedicated to PostgreSQL and one `VMSeg` for DuckDB. This configuration is intentionally sized to reflect realistic constraints in single-server deployments, while still demonstrating the effectiveness of Vista's design. Notably, the memory footprint is not fixed: the system can be scaled to support higher-throughput workloads by provisioning additional `VMSegs` as needed. Baseline systems are configured according to best practices, with their shared buffer pools set to consume 25–40% of total system memory.

5.2 Isolation–Freshness Tradeoff

To explore how Vista operates along the isolation–freshness spectrum, we treat sealing as an explicit *freshness knob* and quantify how its configuration affects both visibility lag and OLTP–OLAP interference. We first (i) characterize Vista's *natural* sealing behavior, i.e., the sealing interval

induced by capacity-based sealing, under varying OLTP load and access skew. We then (ii) quantify how introducing an explicit sealing timeout shifts this behavior, measuring the resulting changes in throughput and sealing frequency. Finally, (iii) using the HATtrick benchmark, we show the throughput frontier and freshness metrics.

5.2.1 Experimental setup. We first characterize sealing intervals and then perform a tradeoff sweep by varying the sealing timeout.

(i) *Sealing-interval characterization.* We run pure OLTP workloads (TPC-C and HyBench OLTP) under a capacity-only sealing policy (timeout disabled), varying the client count (1–32) and access pattern (uniform vs. skewed). We report the mean sealing-to-sealing interval, averaged over 30 consecutive sealing events.

(ii) *Tradeoff sweep with timeouts.* We run 5-minute HTAP workloads (CH-benCHmark and HyBench) with 16-thread DuckDB while varying the sealing timeout (No Timeout, 1 s, 500 ms, 250 ms, 100 ms, (only for CH-benCHmark) 50 ms). For each timeout, we evaluate two OLTP client settings (4 and 12) and two access patterns (uniform vs. skewed), and report transaction throughput (TPM), analytical throughput (QPM), and the observed sealing-to-sealing interval.

(iii) *Holistic benchmarking*. We run HATtrick with a 1-minute warm-up followed by a 10-minute measurement period. For each system, we report two metrics: (i) a *throughput frontier*, defined as the skyline of observed transactional throughput versus analytical throughput, and (ii) a *freshness score*, which captures how quickly analytical queries incorporate recent updates. For Vista, we additionally sweep the sealing timeout (10 s, 1 s, 100 ms) to shift where the system operates along the isolation–freshness spectrum.

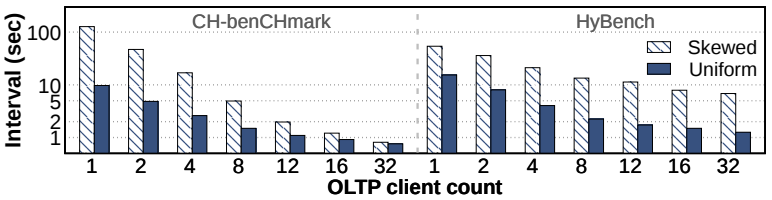


Fig. 8. Mean sealing interval across varying OLTP loads.

5.2.2 Sealing-interval characteristics. Figure 8 shows that, without time-based sealing, the sealing interval depends on both workload intensity and access pattern. We observe a clear inverse relationship between OLTP concurrency and the sealing interval: as the number of OLTP clients increases, buffer allocation accelerates, and sealing is triggered more frequently. At high concurrency (32 clients), the interval drops to sub-second levels in CH-benCHmark. Access skew further widens the gap: under *skewed* access, transactions repeatedly access a small working set, which slows segment filling and therefore delays sealing. In the extreme (single client), this can postpone sealing—and thus the exposure of recent updates to OLAP—by over 100 seconds.

5.2.3 Performance isolation under sealing timeouts. Figure 9 quantifies how the *freshness knob* (time-based sealing) perturbs performance isolation, reporting OLTP throughput (TPM; bars) and geometric mean of OLAP query latency (sec; line) as we sweep the sealing timeout. Intuitively, tighter timeouts bound visibility lag by forcing more frequent generation turnovers, but they also increase the rate of sealing-related work (segment transitions, background flush, and *Copy-on-Seal* on updates that hit sealed segments). The key question is therefore whether a tighter freshness bound comes at a noticeable cost in OLTP–OLAP interference.

CH-benCHmark results. Under *skewed* access with 12 OLTP clients, reducing the timeout from `No Timeout` to 50 ms decreases OLTP throughput (18%), while average OLAP query latency

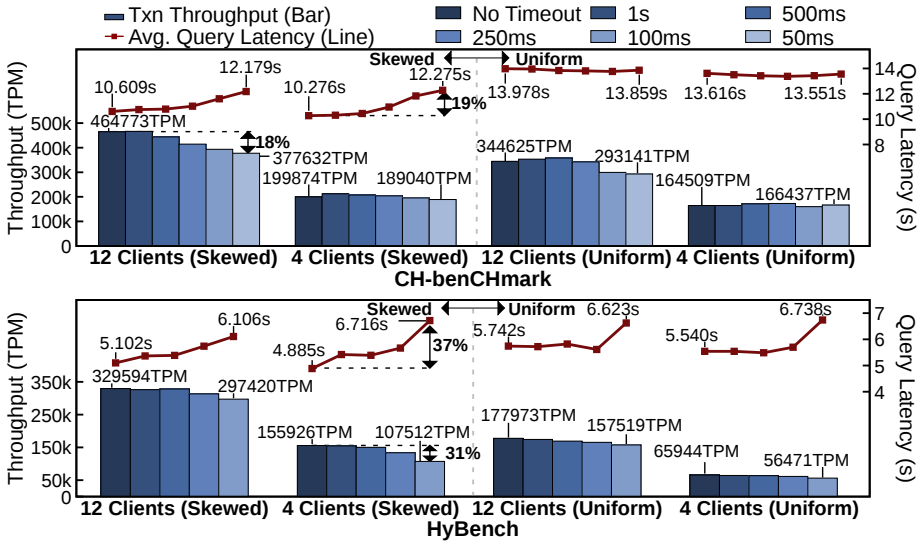


Fig. 9. Performance isolation under varying sealing timeouts.

increases (20%). With fewer writers (4 clients), the throughput drop is smaller. Skew concentrates updates on a small working set and fills segments slowly, so aggressive timeouts introduce *extra* sealing/writeback cycles that would not occur under capacity-only sealing. Under *uniform* access, the OLAP latency curve is largely flat across timeouts, and the throughput impact is more modest and primarily visible under very aggressive timeouts. In this region, segments fill quickly even without timeouts, so the timeout sweep changes the sealing frequency less dramatically; consequently, additional sealing-induced interference is limited.

HyBench results. HyBench exhibits the same qualitative pattern: tighter timeouts reduce OLTP throughput (31%) and increase OLAP latency (37%), with the largest degradation under *skewed* access and lower OLTP concurrency. Under higher concurrency and/or uniform access, the curves are flatter, indicating that moderate timeouts can be applied with smaller isolation penalties.

Takeaway. Overall, time-based sealing provides a practical lever to bound visibility lag, but its cost is workload dependent. In particular, *aggressive* timeouts (tens of ms) can undermine performance isolation under low-write or highly skewed workloads, whereas *moderate* timeouts (hundreds of ms to seconds) typically shift the operating point with substantially smaller interference.

5.2.4 Throughput frontier and freshness score. Figure 10 maps Vista's isolation landscape as a TPS–QPS *throughput frontier* under HATtrick while sweeping the sealing timeout, and includes frontiers for PG(pg_duckdb) and PG+D for comparison. The frontier shape remains largely stable across timeouts, suggesting that time-based sealing can be used as a low-overhead *freshness knob* over this range. HATtrick also reports freshness directly: with a loose 10 s timeout (close to `No Timeout`), Vista achieves a freshness score of 2.08 s; tightening the timeout to 1 s improves the score to 0.85 s; and an aggressive 100 ms timeout yields a sub-second score of 85 ms. Overall, these results indicate that Vista can trade a small fraction of throughput for shorter visibility lag, allowing users to select an operating point on the isolation–freshness spectrum. Unless stated otherwise, we therefore use the default capacity-based sealing in the remaining experiments to reflect Vista's default configuration.

One notable behavior in Figure 10 is Vista's high transactional throughput in the OLTP-only region (the x-axis intercept). This advantage stems from the interaction between Vista's generational buffer management and HATtrick's transactional mix, which is dominated by simple, append-only

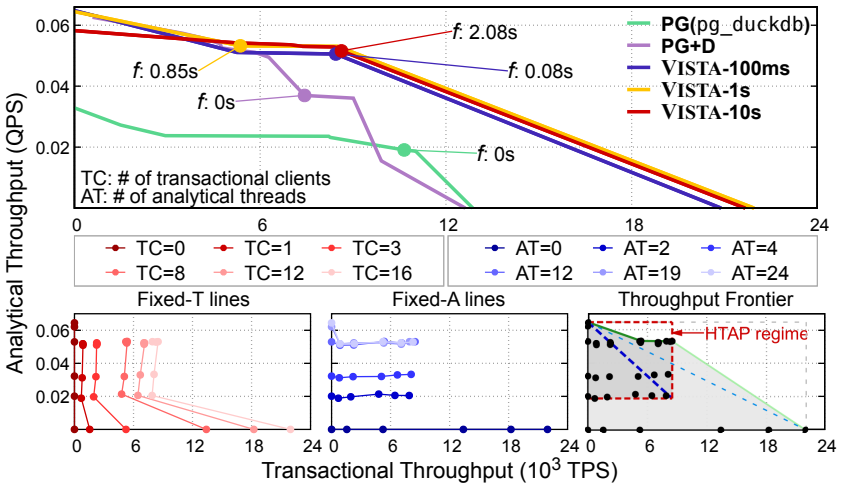


Fig. 10. Throughput frontier and freshness score (HATtrick benchmark).

INSERTs with high *temporal locality*. In this setting, VISTA's sequential allocation avoids buffer-replacement overheads, and *Copy-on-Seal* effectively carries hot pages across generations. Setting aside this OLTP-only peak, the HTAP frontiers with non-zero analytical clients remain close to the ideal bounding box (red dotted line), indicating that VISTA preserves performance isolation under HATtrick mixed loads.

A further observation is that PG+D and PG(pg_duckdb) report a freshness score of 0 (i.e., queries always observe the latest committed state), even though PG+D copies data from PostgreSQL via COPY, whereas VISTA exhibits a non-zero freshness score. This may appear counterintuitive: why does the data-movement tax not emerge as a non-zero freshness score? As discussed in the literature [39], the discrepancy stems from HATtrick's definition of freshness, which differs from the commonly used notion of *visibility delay*. In PG+D, HATtrick marks the COPY phase as the beginning of the query; consequently, the copying delay is absorbed into query latency rather than reported as explicit visibility lag in the freshness score. In contrast, VISTA's sealing policy is decoupled from query start times, so the resulting visibility lag is treated as an explicit *(time) lag* [37, 43] and reported as a non-zero freshness score.

5.3 Performance Isolation

To evaluate performance isolation, we examine whether OLAP query latency remains stable as the number of OLTP workers increases in VISTA.

5.3.1 Experimental setup. To evaluate the degree of performance isolation, we executed the CH-benCHmark and HyBench queries as the OLAP workload with 32-thread DuckDB, while varying the number of concurrent OLTP workers issuing the benchmarks' respective OLTP transactions. We quantify performance isolation by measuring the *per-query latency* of OLAP execution, as this metric directly reflects the degree to which OLTP interference impacts analytical responsiveness. For CH-benCHmark, latency is measured under three configurations: *OLAP-only* (no concurrent OLTP), *OLAP with 4 OLTP workers*, and *OLAP with 12 OLTP workers*. For HyBench, latency is measured under two configurations: *OLAP-only* and *OLAP with 16 OLTP workers*. To highlight the effect of OLTP contention, all latency measurements are normalized against the 12-OLTP-worker (CH-benCHmark) and the 16-OLTP-worker (HyBench) configuration; they are collected after the system has reached a steady state to eliminate cold-start artifacts. To assess the performance

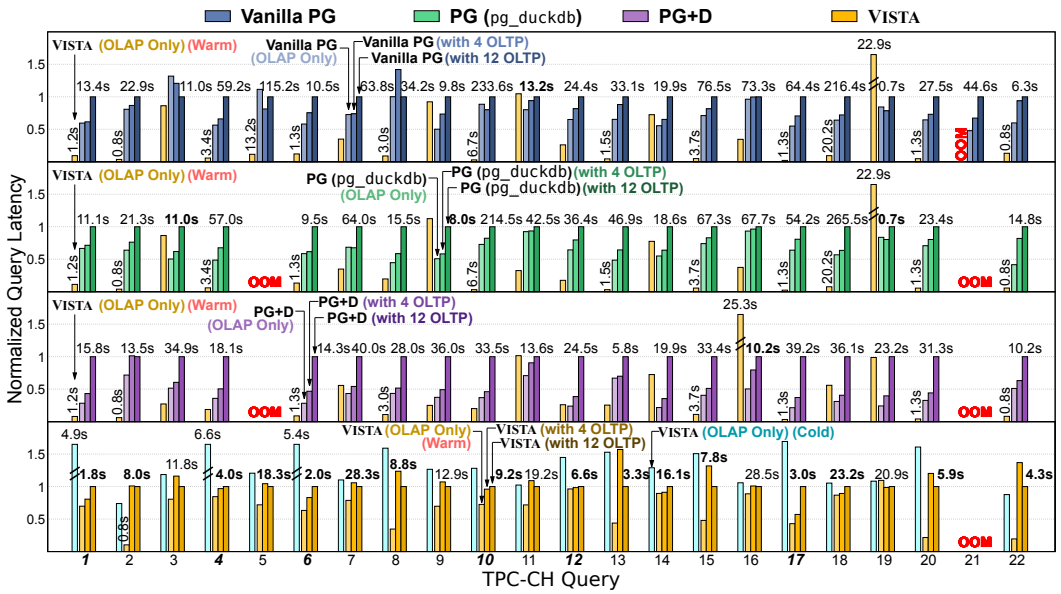


Fig. 11. OLAP performance isolation (CH-benCHmark): the latency of OLAP queries under different levels of OLTP workloads. Numbers above bars indicate query latency (sec) with 12 OLTP workers.

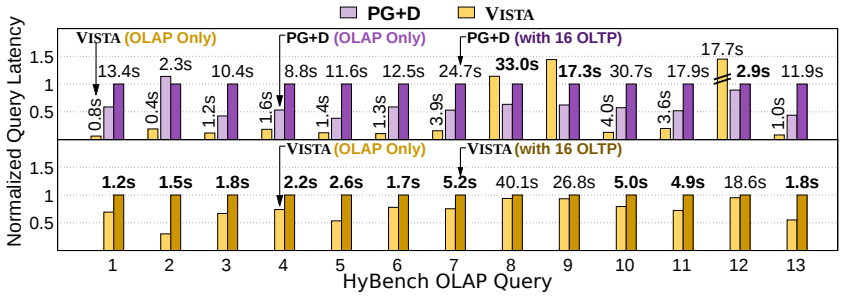


Fig. 12. OLAP performance isolation (HyBench)

isolation capabilities of VISTA, we compare it against three baselines: (1) Vanilla PG, (2) PostgreSQL (pg_duckdb), and (3) PG+D.

5.3.2 Performance isolation results. Figure 11 demonstrates that VISTA delivers near-constant OLAP performance even as the OLTP load scales, while all three baseline systems suffer substantial latency inflation under contention. Notably, when increasing the number of OLTP workers from 4 to 12, Vanilla PG experiences a maximum latency increase of 60.3 seconds (Q18), PG(pg_duckdb) spikes by up to 96.6 seconds (Q18), and PG+D by 24.6 seconds (Q17). In contrast, VISTA limits its worst-case increase to just 2.4 seconds (Q18), clearly showcasing its strong OLAP performance isolation. Although a modest performance gap exists between the OLAP-only and OLAP+OLTP cases in VISTA, this discrepancy is not due to interference from OLTP activity. Rather, it stems from format cache invalidation, as analyzed in section 5.3.3. Because the OLTP engine modifies pages in memory, corresponding cache entries on the OLAP side must be conservatively invalidated to ensure correctness—introducing recomputation overhead.

Under the 12-OLTP-worker configuration—a **highly contended scenario**—VISTA achieves **substantial speedups** over PG+D (the strongest among all baselines), with notable gains in

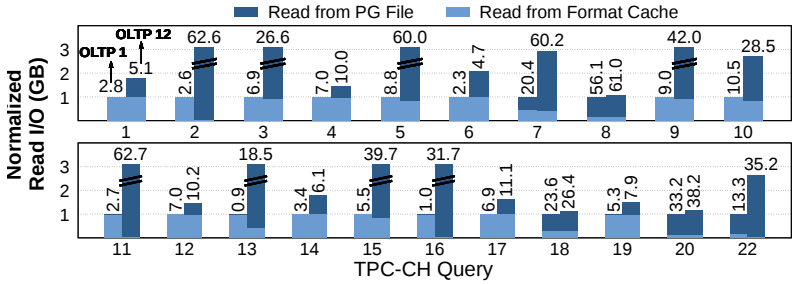


Fig. 13. Normalized read I/O with the internal breakdown.

queries such as Q1(8.81×), Q4(4.53×), Q6(7.17×), Q10(3.62×), Q12(3.70×), and Q17(12.88×) (see Figure 11). These queries primarily operate on tables with *append-only-like* access patterns—such as `orders` and `orderline`—where new records are frequently inserted but most old records are rarely updated. For such workloads, Vista's format cache proves highly effective in reducing redundant page reads and conversion costs, significantly lowering overall I/O volume. This explains the substantial performance advantage Vista maintains across OLAP queries under concurrent transactional pressure.

We focus our HyBench analysis on PG+D, as the CH-benCHmark results established it as the dominant baseline. Figure 12 confirms that Vista maintains consistent performance trends: while PG+D suffers from OLAP latency inflation of up to 13.0 seconds (Q10), Vista exhibits a maximum latency increase of only 2.4 seconds (Q8). These results demonstrate that Vista's performance isolation remains robust across diverse workload characteristics.

5.3.3 The impact of the format cache. To further isolate the impact of the format cache, we evaluate a configuration of Vista with caching disabled and report its performance in the bottom row of Figure 11, measured under OLAP-only conditions. This comparison highlights the critical role of format reuse in maintaining scan efficiency, even when OLTP activity is absent. We further analyze the interaction between format caching and OLTP load by comparing performance under two representative contention levels: low (1 OLTP worker) and high (12 OLTP workers). As shown in Figure 13, the benefits of caching diminish as OLTP load increases, due to frequent invalidations of cached segments. Under high contention, the OLAP-side Vista runtime is forced to fall back on raw PostgreSQL pages, re-executing format conversion and regenerating cache files. Nevertheless, Vista sustains consistent OLAP latency even under frequent invalidation, owing to its fine-grained cache management and prefetching policy.

5.4 Dynamic Workloads

To evaluate Vista's ability to sustain high-throughput transactional processing alongside low-latency analytical queries, we conduct a time-phased experiment that stresses the system under dynamically shifting translytical workloads. This test examines the runtime interaction between OLTP and OLAP tasks and quantifies their mutual interference.

5.4.1 Experimental setup. We run a 15-minute benchmark in three 300-second phases: (i) an initial baseline phase with an isolated workload (either OLTP-only or OLAP-only), (ii) a mixed-workload phase where the complementary workload is introduced, (iii) a final recovery phase after the interfering workload is removed. This setting enables a controlled evaluation of interference and recovery behaviors in a dynamic setting. For OLTP, we measure transaction throughput (TPM) with 12 workers, sampled at 20-second intervals using the HammerDB counter. For OLAP, we monitor the per-query latency of TPC-CH queries, except for Q5 and Q21, which caused an OOM error for more than one system. We also use Linux's `iostat` to track disk read and write bandwidth

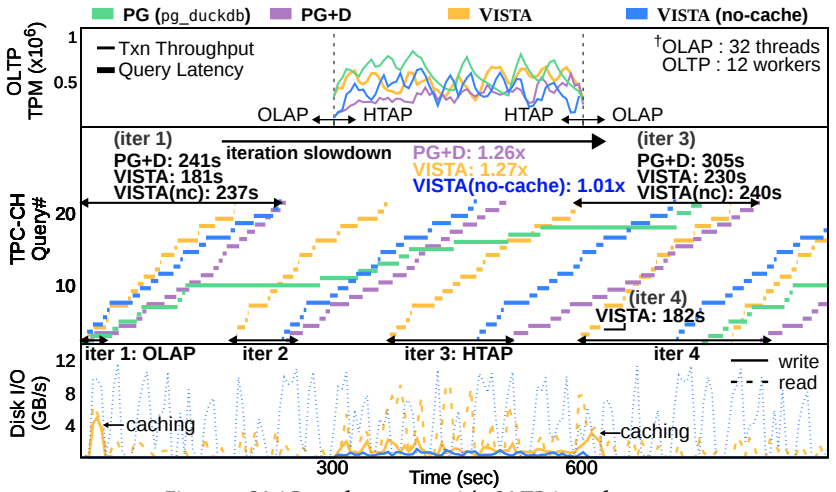


Fig. 14. OLAP performance with OLTP interference.

(in GB/s) to characterize the resource utilization during each phase. All experimental runs were conducted after a one-minute warm-up phase.

In the OLAP-focused evaluation, we exclude Vanilla PG as it lacks a dedicated analytical engine. Instead, we introduce Vista's *no-cache* variant, which disables both reading and writing of the *format cache*, thereby isolating the impact of resource contention from that of cache invalidation. This configuration allows us to attribute OLAP performance differences solely to interference effects. Conversely, in the OLTP-focused experiments, we limit DuckDB to 16 threads to reserve sufficient CPU resources for the OLTP engine on our 32-core machine setting. Also, we enable checkpointing in all other baseline systems to mirror Vista's implicit checkpointing behavior via generational flushing. Specifically, we configure `max_wal_size` to 1 GB (default) and set `checkpoint_timeout` to 30 seconds to trigger periodic checkpointing. This alignment ensures that all systems are subject to comparable background I/O pressures during transactional processing.

5.4.2 HTAP performance results. Evaluation results—Figure 14 and Figure 15—confirm that Vista maintains strong OLAP performance isolation even under dynamic HTAP conditions. In Figure 14, the Vista (no-cache) variant incurs only a negligible slowdown of 1.01× between the OLAP-only and HTAP phases, isolating the impact of pure resource contention. In comparison, the standard Vista configuration experiences a modest 1.27× slowdown, which can thus be attributed primarily to *format cache* invalidation rather than interference. The only baseline with comparable capabilities, PG+D, exhibits a similar 1.26× slowdown, entirely due to contention effects. Notably, Vista recovers immediately following the termination of the OLTP workload—its performance in the fourth OLAP iteration matches that of the first, demonstrating resilience and low-latency recovery. Disk I/O statistics further reveal that the Vista (no-cache) variant fully saturates the PCIe 5.0 NVMe SSD's read bandwidth, demonstrating its ability to exploit high-performance storage devices. In contrast, the standard Vista instance leverages its column-selective cache reload policy: it proactively repopulates the format cache during the initial stages of the pure OLAP phases (0–300s, 600–900s) and subsequently suppresses redundant I/O, achieving better efficiency across repeated scans.

Figure 15 shows that Vista achieves the highest OLTP throughput during the HTAP phase, underscoring its strong performance isolation—even in the presence of background generational segment flushing. Since all systems share the same OLTP engine, this advantage is a direct result of Vista's physically separated buffer segments, which minimize interference from concurrent analytical workloads. Nevertheless, two factors influence overall system behavior. First, unavoidable

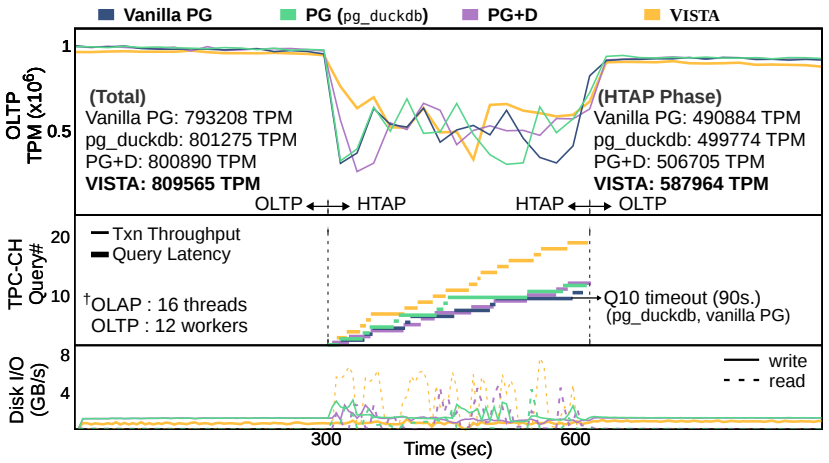


Fig. 15. OLTP performance with OLAP interference.

storage-level contention contributes to a throughput dip during the HTAP phase: the OLAP engine issues heavy read traffic, leading to increased I/O pressure that affects all workloads during the HTAP phase (see **Disk I/O**). Second, a slight but consistent throughput drop is observed even during OLTP-only phases. This stems from the generational flush mechanism, where sealing a VMSeg, performing *Copy-on-Seal* operations, and remapping the segment to the OLAP engine incur measurable overhead. Importantly, this cost is tunable. In our setup, we allocate a single 1 GB OLTP segment. Increasing the number of VMSegs would reduce flush frequency and amortize remapping overheads, thereby improving OLTP throughput without compromising isolation.

5.5 In-depth Analysis of the Vista Runtime

To quantify Vista's OLAP-side runtime overhead, we performed an in-depth analysis using CH-benCHmark queries. To isolate the OLAP path, we disable the OLTP engine so no remapping events occur; thus, all accesses are served via storage prefetching and format-aware caching. We use the same configurations as in Table 1.

5.5.1 Experimental setup. We evaluate two cache conditions: *cold*, where no format cache exists at query start, and *warm*, where all format caches are pre-populated. We exclude Q21 due to an out-of-memory failure from excessive intermediate materialization. To quantify runtime overheads, we instrument Vista to break down per-query CPU time into eight components: (i) **PG Reading** (reading row-store pages without cache), (ii) **Format Converting** (row-to-column translation), (iii) **Visibility Checking** (per-tuple MVCC filtering), (iv) **Cache Serializing**, (v) **Cache Writing**, (vi) **Cache Reading**, (vii) **Cache Deserializing**, and (viii) **Query Processing** (vectorized execution). This breakdown contrasts cold vs. warm cache runs and quantifies the conversion–reuse tradeoff.

5.5.2 Evaluation results. Figure 16 shows the per-query CPU-time breakdown of Vista's OLAP runtime for all analytic queries under *cold* and *warm* cache states. We observe:

(1) **Cold-cache runs are conversion-bound.** *Format conversion* (row-to-column) dominates cold execution, accounting for 7.9–47.2% of CPU time, while *visibility checking* (MVCC tuple filtering) contributes 1.0–8.5%. In the warm state, both costs disappear because converted data is served from the format cache.

(2) **Materialization overhead is secondary.** *Cache serialization* and *cache writing* appear only in cold runs and are modest relative to conversion; moreover, they are amortized across queries.

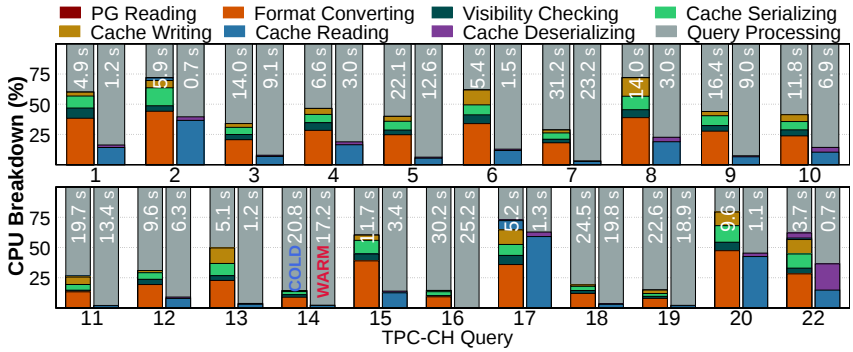


Fig. 16. CPU breakdown under cold and warm caches.

(3) **Warm-cache runs shift cost to query execution.** With a populated cache, data preparation reduces to *cache reading* and *deserialization*, yielding 1.2×–8.4× speedups over cold runs, after which most CPU time is spent in *query processing*.

(4) **Opportunistic intra-query reuse occurs.** Even in cold runs, some queries (e.g., Q2, Q11, Q17, Q22) exhibit non-zero cache I/O due to repeated table accesses (e.g., nested subqueries), leading to cache hits within a single query.

(5) **Sequential heap scans are CPU-cheap; cache reads can be I/O-fragmented.** *PostgreSQL page reading* accounts for $< 0.15\%$ of CPU time in cold runs due to effective sequential prefetching. In contrast, *cache reading* can reach 59.1% because per-column cache objects induce scattered I/O that prefetches less effectively.

6 CONCLUSION

Although HTAP systems have become important for enabling real-time analytics over transactional data, achieving cross-engine HTAP within a single-server architecture remains challenging due to shared resource contention and the difficulty of balancing *performance isolation* with *data freshness*. We present Vista, a DB–OS co-designed buffer-management framework that rethinks the buffer cache around segment-granular ownership and policy-controlled visibility. Rather than claiming to eliminate data staleness or performance interference, Vista makes the isolation–freshness tradeoff explicit by exposing sealing as a tunable mechanism. Across diverse benchmarks, our evaluation characterizes how sealing policies shift the system's operating point, quantifying the resulting visibility lag and OLTP–OLAP interference. Evaluation results show that Vista reduces cross-engine interference compared to shared-buffer and decoupled baselines in our setup, while providing a practical knob to control data freshness for analytics. We hope Vista facilitates DB-OS co-design research in single-node HTAP.

Acknowledgments

The authors sincerely thank the anonymous reviewers for their insightful and constructive feedback, which significantly improved the quality of this work. This research was supported by the National Research Foundation of Korea (NRF) grants funded by the Korean government (MSIT) (No. 2022R1A2C2008427: Next-Generation DBMS Architecture for High-Performance Hybrid Transactional/Analytical Processing and No. RS-202300222663: Center for Optimizing Hyperscale AI Models and Platforms).

References

[1] Amazon Web Services, Inc. 2023. Amazon Aurora zero-ETL Integration with Amazon Redshift. https://aws.amazon.com/rds/aurora/zero-etl/. Accessed: 2025-04-10.

[2] Raja Appuswamy, Goetz Graefe, Renata Borovica-Gajic, and Anastasia Ailamaki. 2019. The five-minute rule 30 years later and its impact on the storage hierarchy. *Commun. ACM* 62, 11 (Oct. 2019), 114–120. doi:10.1145/3318163

[3] Raja Appuswamy, Manos Karpathiotakis, Danica Porobic, and Anastasia Ailamaki. 2017. The Case for Heterogeneous HTAP. In *CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. http://cidrdb.org/cidr2017/papers/p21-appuswamy-cidr17.pdf

[4] AWS Inc. 2025. Compute optimized Amazon EC2 instance types. Available at https://aws.amazon.com/ec2/instance-types/compute-optimized/. Accessed: 2025-10-05.

[5] Michael J. Carey, David J. DeWitt, Michael J. Franklin, Nancy E. Hall, Mark L. McAuliffe, Jeffrey F. Naughton, Daniel T. Schuh, Marvin H. Solomon, C. K. Tan, Odysseas G. Tsatalos, Seth J. White, and Michael J. Zwilling. 1994. Shoring up persistent applications. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data* (Minneapolis, Minnesota, USA) *(SIGMOD '94)*. Association for Computing Machinery, New York, NY, USA, 383–394. doi:10.1145/191839.191915

[6] Citus Data. 2020. Tools for running CH-benCHmark with HammerDB. Available at https://github.com/citusdata/ch-benchmark. Accessed: 2025-10-05.

[7] Richard Cole, Florian Funke, Leo Giakoumakis, Wey Guy, Alfons Kemper, Stefan Krompass, Harumi Kuno, Raghunath Nambiar, Thomas Neumann, Meikel Poess, Kai-Uwe Sattler, Michael Seibold, Eric Simon, and Florian Waas. 2011. The Mixed Workload CH-BenCHmark. In *Proceedings of the Fourth International Workshop on Testing Database Systems* (Athens, Greece) *(DBTest '11)*. Association for Computing Machinery, New York, NY, USA, Article 8, 6 pages. doi:10.1145/1988842.1988850

[8] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, Allison W. Lee, Ashish Motivala, Abdul Q. Munir, Steven Pelley, Peter Povinec, Greg Rahn, Spyridon Triantafyllis, and Philipp Unterbrunner. 2016. The Snowflake Elastic Data Warehouse. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. Association for Computing Machinery, New York, NY, USA, 215–226. doi:10.1145/2882903.2903741

[9] Google Inc. 2022. Google AlloyDB for PostgreSQL. https://cloud.google.com/products/alloydb. Accessed: 2025-04-10.

[10] Goetz Graefe. 2008. The Five-Minute Rule 20 Years Later: and How Flash Memory Changes the Rules: The old rule continues to evolve, while flash memory adds two new rules. *Queue* 6, 4 (July 2008), 40–52. doi:10.1145/1413254.1413264

[11] Jim Gray and Goetz Graefe. 1997. The five-minute rule ten years later, and other computer storage rules of thumb. *SIGMOD Rec.* 26, 4 (Dec. 1997), 63–68. doi:10.1145/271074.271094

[12] Jim Gray and Gianfranco Putzolu. 1987. The five-minute rule for trading memory for disk accesses. *ACM SIGMOD Record* 16, 3 (1987), 395–398.

[13] The PostgreSQL Global Development Group. 2026. Database Page Layout. https://www.postgresql.org/docs/current/storage-page-layout.html#id-1.10.18.8.2.2. Accessed: 2026-1-17.

[14] HammerDB Version 4.4. 2022. Available at https://github.com/TPC-Council/HammerDB/releases/tag/v4.4. Accessed: 2025-10-05.

[15] Dongxu Huang, Qi Liu, Qiu Cui, Zhuhe Fang, Xiaoyu Ma, Fei Xu, Li Shen, Liu Tang, Yuxing Zhou, Menglong Huang, Wan Wei, Cong Liu, Jian Zhang, Jianjun Li, Xuelian Wu, Lingyu Song, Ruoxi Sun, Shuaipeng Yu, Lei Zhao, Nicholas Cameron, Liquan Pei, and Xin Tang. 2020. TiDB: A Raft-Based HTAP Database. *Proc. VLDB Endow.* 13, 12 (aug 2020), 3072–3084. doi:10.14778/3415478.3415535

[16] Alfons Kemper and Thomas Neumann. 2011. HyPer: A hybrid OLTP&OLAP main memory database system based on virtual memory snapshots. In *2011 IEEE 27th International Conference on Data Engineering*. 195–206. doi:10.1109/ICDE.2011.5767867

[17] Ralph Kimball and Margy Ross. 2002. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling* (2nd ed.). John Wiley & Sons, Inc., USA.

[18] Per-Åke Larson, Adrian Birka, Eric N. Hanson, Weiyun Huang, Michal Nowakiewicz, and Vassilis Papadimos. 2015. Real-Time Analytical Processing with SQL Server. *Proc. VLDB Endow.* 8, 12 (aug 2015), 1740–1751. doi:10.14778/2824032.2824071

[19] Viktor Leis, Alfons Kemper, and Thomas Neumann. 2018. LeanStore: In-Memory Data Management beyond Main Memory. In *Proceedings of the 2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 185–196. doi:10.1109/ICDE.2018.00024

[20] Guoliang Li and Chao Zhang. 2022. HTAP Databases: What is New and What is Next. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 2483–2488. doi:10.1145/3514221.3522565

[21] Darko Makreshanski, Jana Giceva, Claude Barthels, and Gustavo Alonso. 2017. BatchDB: Efficient Isolated Execution of Hybrid OLTP+OLAP Workloads for Interactive Applications. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 37–50. doi:10.1145/3035918.3035959

[22] Norman May, Alexander Böhm, Daniel Ritter, Frank Renkes, Mihnea Andrei, and Wolfgang Lehner. 2025. SAP HANA Cloud: Data Management for Modern Enterprise Applications. In *Companion of the 2025 International Conference on Management of Data* (Berlin, Germany) *(SIGMOD/PODS '25)*. Association for Computing Machinery, New York, NY, USA, 580–592. doi:10.1145/3722212.3724452

[23] Microsoft. 2025. Heaps (tables without clustered indexes). https://learn.microsoft.com/en-us/sql/relational-databases/indexes/heaps-tables-without-clustered-indexes?view=sql-server-ver17. Accessed: 2026-1-17.

[24] Microsoft Cooperation. 2025. Microsoft Azure Virtual Machines: Fsv2 sizes series. Available at https://learn.microsoft.com/en-us/azure/virtual-machines/sizes/compute-optimized/fsv2-series?tabs=sizebasic#sizes-in-series. Accessed: 2025-10-05.

[25] Elena Milkai, Yannis Chronis, Kevin P. Gaffney, Zhihan Guo, Jignesh M. Patel, and Xiangyao Yu. 2022. How Good is My HTAP System?. In *Proceedings of the 2022 International Conference on Management of Data* (Philadelphia, PA, USA) *(SIGMOD '22)*. Association for Computing Machinery, New York, NY, USA, 1810–1824. doi:10.1145/3514221.3526148

[26] Elena Milkai, Xiangyao Yu, and Jignesh M. Patel. 2025. Hermes: Off-the-Shelf Real-Time Transactional Analytics. *Proc. VLDB Endow.* 18, 8 (Sept. 2025), 2334–2347. doi:10.14778/3742728.3742731

[27] Patrick O'Neil, Elizabeth O'Neil, Xuedong Chen, and Stephen Revilak. 2009. The Star Schema Benchmark and Augmented Fact Table Indexing. In *Performance Evaluation and Benchmarking*, Raghunath Nambiar and Meikel Poess (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 237–252.

[28] Oracle. 2025. Tables and Table Clusters. https://docs.oracle.com/en/database/oracle/oracle-database/21/cncpt/tables-and-table-clusters.html#GUID-F845B1A7-71E3-4312-B66D-BC16C198ECE5. Accessed: 2026-1-17.

[29] Oracle. 2026. The MyISAM Storage Engine. https://dev.mysql.com/doc/refman/8.4/en/myisam-storage-engine.html. Accessed: 2026-1-17.

[30] Oracle Corporation and/or its affiliates. 2022. MySQL HeatWave: One MySQL Database for OLTP, OLAP, and Machine Learning. Available at https://www.oracle.com/a/ocom/docs/mysql-database-service-with-heatwave.pdf.

[31] Fatma Özcan, Yuanyuan Tian, and Pinar Tözün. 2017. Hybrid Transactional/Analytical Processing: A Survey. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 1771–1775. doi:10.1145/3035918.3054784

[32] Iraklis Psaroudakis, Florian Wolf, Norman May, Thomas Neumann, Alexander Böhm, Anastasia Ailamaki, and Kai-Uwe Sattler. 2014. Scaling Up Mixed Workloads: A Battle of Data Freshness, Flexibility, and Scheduling. In *Performance Characterization and Benchmarking. Traditional to Big Data - 6th TPC Technology Conference, TPCTC 2014, Hangzhou, China, September 1-5, 2014. Revised Selected Papers (Lecture Notes in Computer Science, Vol. 8904)*, Raghunath Nambiar and Meikel Poess (Eds.). Springer, 97–112. doi:10.1007/978-3-319-15350-6_7

[33] Rajesh Sukhramani. 2025. Oracle Database Insider Blog: Oracle Named a Leader in The Forrester Wave™: Translytical Data Platforms, Q4 2024. Available at https://blogs.oracle.com/database/post/oracle-named-a-leader-in-the-forrester-wave-translytical-data-platforms-q4-2024.

[34] Tobias Schmidt, Dominik Durner, Viktor Leis, and Thomas Neumann. 2024. Two Birds With One Stone: Designing a Hybrid Cloud Storage Engine for HTAP. *Proc. VLDB Endow.* 17, 11 (July 2024), 3290–3303. doi:10.14778/3681954.3682001

[35] Felix Martin Schuhknecht, Jens Dittrich, and Ankur Sharma. 2016. RUMA has it: rewired user-space memory access is possible! *Proc. VLDB Endow.* 9, 10 (June 2016), 768–779. doi:10.14778/2977797.2977803

[36] Ankur Sharma, Felix Martin Schuhknecht, and Jens Dittrich. 2018. Accelerating Analytical Processing in MVCC using Fine-Granular High-Frequency Virtual Snapshotting. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 245–258. doi:10.1145/3183713.3196904

[37] Sijie Shen, Rong Chen, Haibo Chen, and Binyu Zang. 2021. Retrofitting High Availability Mechanism to Tame Hybrid Transaction/Analytical Processing. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, 219–238. https://www.usenix.org/conference/osdi21/presentation/shen

[38] Jeff Shute, Radek Vingralek, Bart Samwel, Ben Handy, Chad Whipkey, Eric Rollins, Mircea Oancea, Kyle Littlefield, David Menestrina, Stephan Ellner, John Cieslewicz, Ian Rae, Traian Stancescu, and Himani Apte. 2013. F1: a distributed SQL database that scales. *Proc. VLDB Endow.* 6, 11 (Aug. 2013), 1068–1079. doi:10.14778/2536222.2536232

[39] Haoze Song, Wenchao Zhou, Heming Cui, Xiang Peng, and Feifei Li. 2024. A survey on hybrid transactional and analytical processing. *The VLDB Journal* 33, 5 (June 2024), 1485–1515. doi:10.1007/s00778-024-00858-9

[40] Sujata Narayana. 2025. Microsoft Power BI Updates Blog: Translytical Task Flows (Preview). Available at https://powerbi.microsoft.com/en-us/blog/announcing-translytical-task-flows-preview/.

[41] The DuckDB Foundation. 2025. DuckDB Postgres extension. Available at https://github.com/duckdb/duckdb-postgres. Accessed: 2025-10-05.

[42] The DuckDB Foundation. 2025. `pg_duckdb v1.0.0`: Official PostgreSQL Extension for DuckDB. Available at https://github.com/duckdb/pg_duckdb. Accessed: 2025-10-05.

[43] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. 2017. Amazon Aurora: Design Considerations for High Throughput Cloud-Native Relational Databases. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 1041–1052. doi:10.1145/3035918.3056101

[44] Jiani Yang, Sai Wu, Yong Wang, Dongxiang Zhang, Yifei Liu, Xiu Tang, and Gang Chen. 2025. Twisted Twin: A Collaborative and Competitive Memory Management Approach in HTAP Systems. *Proc. VLDB Endow.* 18, 10 (Sept. 2025), 3312–3325. doi:10.14778/3748191.3748197

[45] Noel Yuhanna. 2024. *The Forrester Wave™: Translytical Data Platforms, Q4 2024*. Research Report. Forrester Research. https://reprint.forrester.com/reports/the-forrester-wavetm-translytical-data-platforms-q4-2024-90b043e6/index.html Available through Forrester Research subscription or reprint services.

[46] Chao Zhang, Guoliang Li, and Tao Lv. 2024. HyBench: A New Benchmark for HTAP Databases. *Proc. VLDB Endow.* 17, 5 (Jan. 2024), 939–951. doi:10.14778/3641204.3641206

Received October 2025; revised January 2026; accepted Feburary 2026