A Unified Bandwidth Orchestration Framework for Hierarchical Data Storage Systems

Ji Zhang*
Huawei Technologies
Switzerland
dr.jizhang@huawei.com

Li Liu
Wuhan Textile University
China
l_liu@wtu.edu.cn

André Brinkmann
Saarland University
Germany
andre.brinkmann@uni-saarland.de

Giovanni Cherubini
Huawei Technologies
Switzerland
giovanni.cherubini1@huawei.com

Zhu Xiangyu
Huawei Technologies
China
zhuxiangyu11@huawei.com

Shai Bergman
Huawei Technologies
Switzerland
shai.aviram.bergman@huawei.com

Ke Zhou
Huazhong University of Science and Technology
China
zhke@hust.edu.cn

ABSTRACT

Hierarchical Data Storage Systems (HSSs) provide a cost-effective architecture that balances capacity and performance through internal data migration. Prior work has primarily focused on optimizing individual migration tasks, either within or across HSS tiers, or on exploiting device bandwidth to improve overall throughput. These approaches treat migration tasks in isolation, and the performance implications of executing heterogeneous migration tasks concurrently remain largely unexplored despite their prevalence in real-world HSS deployments. The growing adoption of Direct Data Access (DDA) architectures, in which accelerators access storage without CPU mediation, further amplifies this problem by removing a natural bandwidth arbiter from the I/O path.

This paper presents an in-depth analysis of data migration behavior in commercial HSSs, uncovering substantial performance variability when multiple migration tasks execute concurrently. To mitigate this issue, we propose PASCAL, a system-level bandwidth orchestration framework that improves performance robustness in production-grade HSSs. Inspired by hydraulic systems, PASCAL adapts pressure/backpressure-style coordination to the multi-task migration setting: it treats each tier as a pressurized vessel and uses pressure gradients to allocate bandwidth across cache flush, tiering, garbage collection, and DDA flows. We evaluate PASCAL on a commercial OceanStor HSS across three hardware configurations and eight workloads spanning database, AI training, AI inference, and production traces. PASCAL achieves up to 20% higher throughput, 67% lower tail latency, and 79% reduced throughput jitter compared to local state-of-the-art controllers, while also stabilizing the performance jitter introduced by DDA architectures.

PVLDB Reference Format:
Ji Zhang, Li Liu, André Brinkmann, Giovanni Cherubini, Zhu Xiangyu, Shai Bergman, and Ke Zhou. A Unified Bandwidth Orchestration Framework for Hierarchical Data Storage Systems. PVLDB, 19(12): 4209 - 4222, 2026.

*Corresponding author

doi:10.14778/3827998.3828027

1 INTRODUCTION

Hierarchical data storage systems (HSSs) integrate multiple storage tiers, such as NVRAM, SSDs, and HDDs, to balance cost, capacity, and performance [5, 27, 37, 40, 50, 51] (Fig. 1). HSSs rely on continuous background data migrations: inter-tier promotion and eviction, and intra-tier garbage collection, to keep hot data on fast tiers and manage capacity. However, database systems, AI training, real-time analytics, and hyperscaler workloads are increasingly pushing HSSs to their limits [7, 9, 12, 17, 36, 43, 47], making it difficult to sustain robust, predictable performance. This challenge is compounded by the growing trend of co-locating diverse workloads, such as latency-sensitive OLTP transactions, throughput-intensive analytics, and bursty AI training pipelines, on a shared storage pool. In such mixed environments, operators require not only high peak throughput but also stable, low-jitter service across all tenants, since a transient bandwidth spike from one workload's background migration can cascade into Quality-of-Service (QoS) violations for another.

Prior research on HSSs primarily focuses on inter-tier data migration policies [11, 37, 49, 51], cross-tier data placement policies [23, 32, 45], and leveraging the bandwidth of devices across storage tiers [33, 41, 44] to enhance system performance. Furthermore, numerous studies focus on data migration policies within single-tier storage, primarily targeting garbage collection strategies optimized for SSDs or log-structured file systems [13, 25, 34, 38]. However, these approaches treat migration tasks in isolation and overlook a critical operational reality: in production HSSs, multiple

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828027

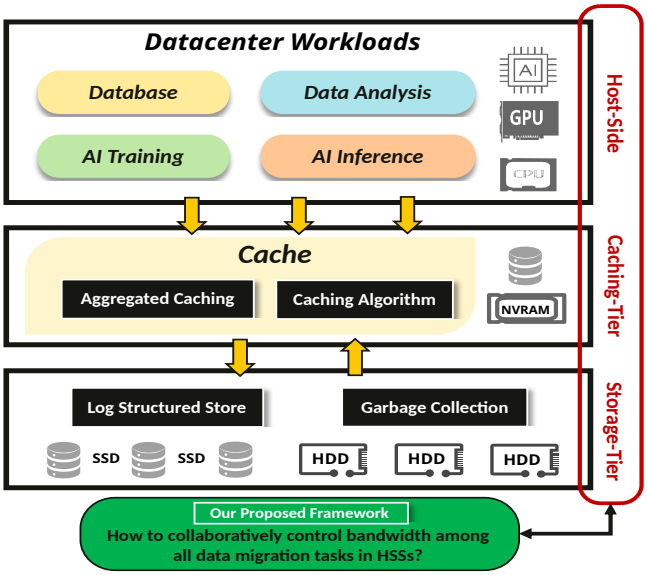


Figure 1: A representative datacenter HSS. Database, analytics, and AI services on the host side share a tiered storage pool composed of an NVRAM cache tier, an SSD capacity tier, and an HDD bulk tier. Foreground client I/O and Direct Data Access (DDA) traffic from accelerators compete with continuous background migration tasks over the same shared storage substrate.

migration tasks execute concurrently, and their interactions can severely destabilize system performance.

The underlying reason is that concurrent migrations share a finite bandwidth limit in each tier. Because these policies are designed in isolation, they cannot anticipate each other's actions. For example, a tiering policy that aggressively promotes data from HDD to SSD may coincide with a file-system garbage-collection cycle on the same SSD tier. Both tasks consume SSD write bandwidth simultaneously: tiering fills the SSD with promoted blocks while GC rewrites live pages to reclaim space, creating a feedback loop of write amplification. The resulting contention spills over to the NVRAM tier, whose eviction path to SSD is now throttled, causing cache back-pressure that inflates foreground read and write latencies. We confirm this effect on a commercial three-tier HSS (see Section 2): combining state-of-the-art migration optimizers yields up to 10% higher throughput but also introduces severe throughput jitter, undermining the QoS guarantees demanded by modern applications [10].

Bandwidth control, regulating *how much* data to migrate per unit time, has proven effective for single tasks such as cache write-back [2, 4, 26, 29, 31, 42, 50]. Extending it to the multi-task HSS setting, however, raises two open challenges: (i) defining a system-wide metric that captures cross-tier migration urgency, and (ii) coordinating bandwidth across heterogeneous tasks (tiering, GC, cache flush) to jointly optimize throughput, jitter, and tail latency. Naïve solutions fall short: static bandwidth partitioning wastes capacity because migrations are inherently phased and episodic, while deploying independent per-task controllers (e.g., one PID or RL agent per migration flow) leads to oscillations, as each controller greedily bids for bandwidth without visibility into the others' actions, reproducing the very interference it aims to eliminate. A viable solution must therefore provide a *global* coordination signal that links all tiers and tasks under a single optimization objective.

To this end, we propose PASCAL, a system-level bandwidth orchestration framework for production-grade HSSs, inspired by *Pascal*'s law[1]. Rather than introducing a new control-theoretic principle, PASCAL adapts pressure/backpressure-style reasoning to a setting where cache flushing, tiering, garbage collection, and Direct Data Access (DDA) traffic were previously controlled independently. PASCAL translates the pressure abstraction into a closed-form bandwidth allocator that operates across heterogeneous tiers and migration tasks, thereby modeling an HSS as a closed-loop hydraulic network (see Fig. 2). Each storage tier is a pressurized vessel, data volumes are the fluid, and inter-tier bandwidths are valves. The hydraulic analogy is a natural fit: just as fluid flows from high-pressure to low-pressure regions until equilibrium is reached, data in an HSS should migrate from congested tiers to underutilized ones, with flow rates proportional to the imbalance. Pressure in PASCAL translates real-time tier utilization into a single dynamic scalar per tier, and pressure differences between tiers directly drive migration bandwidth allocation, yielding a closed-form, interpretable control policy. Because the optimal pressure parameters depend on hardware configuration and workload mix, PASCAL employs Bayesian optimization to tune them online, a sample-efficient strategy well suited to the expensive, real-hardware evaluations inherent in production storage systems.

PASCAL's coordination is especially critical given the emergence of DDA architectures, such as NPU Direct and GPU Direct, that allow accelerators to read and write storage tiers without CPU mediation. DDA removes the CPU as a natural bandwidth arbiter, injecting high-speed, bursty I/O directly into the storage hierarchy. While DDA delivers substantial throughput and latency gains for AI workloads, it also amplifies performance jitter dramatically (up to 2× in our experiments), because the uncoordinated accelerator traffic competes with ongoing background migrations at wire speed. Single-task bandwidth controllers, designed for CPU-mediated I/O, lack the global visibility to manage this contention. PASCAL's system-wide pressure model naturally extends to DDA traffic by treating accelerator I/O as an additional external pressure source.

While the core of PASCAL resides in the storage orchestration layer, its primary objective is to solve a critical QoS challenge: ensuring performance predictability and stability for data management systems and data analytic applications running on HSSs. In modern data centers, database engines interact directly with hierarchical storage. The background migration "noise" from concurrent cache flushes, tier promotions, and garbage collection would surface at the database layer as severe throughput jitter and QoS violations. PASCAL's global orchestration provides a storage-orchestration substrate for stable database execution.

We evaluate PASCAL on a production-grade Huawei OceanStor Dorado 18000 V6 storage system under three hardware configurations that vary the number of tiers and the SSD fleet size, exercising different capacity-to-bandwidth ratios. Eight workloads, ranging

[1]https://www.britannica.com/science/Pascals-principle

from YCSB and Facebook ZippyDB database benchmarks, through MLPerf AI training traces (U-Net3D, ResNet-50, CosmoFlow), to anonymized production traces from a major bank, an LLM training cluster, and a large-scale AI inference platform, stress the system under diverse access patterns and concurrency levels. Experiments also include DDA configurations in which Ascend 910B NPUs bypass the CPU for direct storage access. Across all settings, PASCAL achieves up to 20% higher throughput, 67% lower tail latency, and 79% reduced throughput jitter compared to state-of-the-art baselines, demonstrating that pressure-aware coordination generalizes across workload types, hardware scales, and I/O architectures.

The main contributions of this paper are as follows.

- We conduct an empirical study on a commercial HSS showing that state-of-the-art migration optimizers, when executed concurrently, improve throughput but introduce severe performance jitter.
- We define a *pressure* metric that unifies tier-level utilization and migration urgency across an entire HSS hierarchy, enabling system-wide coordination of heterogeneous data migration tasks.
- We design a pressure-gradient bandwidth allocator, based on Pascal's law, that coordinates multiple concurrent migrations to jointly optimize throughput, jitter, and tail latency.
- We evaluate PASCAL on a commercial three-tier HSS under eight workloads—including database benchmarks, AI training, and production traces—and three hardware configurations. PASCAL achieves up to 20% higher throughput, 67% lower tail latency, and 79% reduced jitter versus state-of-the-art baselines.

2 BACKGROUND AND MOTIVATION

Hierarchical data storage systems: Hierarchical data storage systems (HSSs) are a key building block of modern data infrastructures. By dynamically migrating data across tiers, such as high-speed NVRAM, mid-tier SSDs, and high-capacity HDDs, HSSs optimize performance for diverse workloads while balancing capacity and minimizing operational costs. The left side of Fig. 2 illustrates a canonical three-tier HSS, which can migrate data between tiers on demand. Client I/O requests are serviced from the fastest available tier, while background data migration tasks continuously redistribute data to align with workload patterns.

Data migration tasks: Data migrations within HSSs can change the data distribution between the storage tiers, which affects the capacity utilization of each tier, thereby shaping overall system performance. Based on the source and destination of the data transfer, data migration in HSSs can be classified into three classes:

(1) *External data traffic* is triggered by client applications or direct data access from GPU/NPU.
(2) *Inter-tier migration* focuses on tiering operations where data is moved between tiers, e.g., from NVRAM to SSD.
(3) *Intra-tier migration* refers to single-tier data management operations such as file system garbage collection (GC).

As external data traffic continuously flows into the storage system, storage pressure across all tiers within HSSs steadily increases. To maintain a balance between storage capacity and performance,

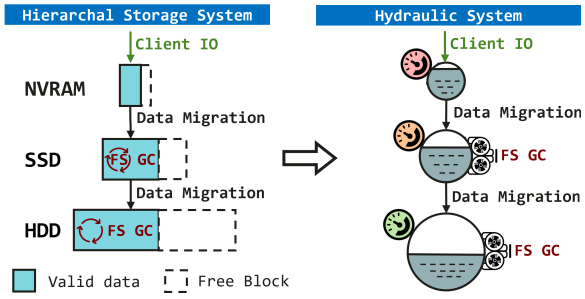


Figure 2: HSS and migration tasks as a hydraulic system. *Left:* the canonical three-tier HSS. *Right:* PASCAL's hydraulic abstraction, where each tier is a pressurized vessel, inter-tier bandwidths are valves, intra-tier garbage collection is a pressure-relief outlet to a free-space sink, and external client/DDA traffic acts as a heat source.

the HSSs inevitably trigger multiple data migration tasks, including intra-tier data migration and inter-tier data migration.

2.1 Analysis

To investigate the impact of concurrent execution of multiple data migration tasks on HSS performance, we conduct extensive experiments on a commercial three-tier HSS (configuration T1 in Section 4), which prioritizes performance stability with built-in heuristic data migration tasks (see "Native" configuration in Section 4). Based on this HSS, we integrate three state-of-the-art optimization techniques to enhance the system performance (i.e., the **"Mixture"** configuration): Ocach [44] for cache traffic optimization, Sibyl [37] for tiering optimization, and ScaleLFS [14] for in-file-system GC optimization. The optimization objectives of the aforementioned methods and their corresponding tiers are shown in Table 1. We experiment with a long-running YCSB benchmark [8], which is commonly used in database benchmarking, and measure the throughput over time.

Fig. 3(a) compares the Mixture and Native configurations on HSS. Compared to Native HSS, Mixture HSS increases average throughput by 10%, but introduces significant performance jitter. The throughput jitter stems from cutting-edge optimization techniques designed to achieve optimal performance for distinct and independent tasks.

Observation #1. Integrating multiple cutting-edge optimization techniques within an HSS significantly enhances performance, but the resulting throughput fluctuations compromise system stability.

To zoom in on the impact of multitask concurrency on HSS performance, we perform a stepwise experiment: in each step, we apply one of the optimizations incrementally. The results are shown in Fig. 3(b). We find that in Mixture HSS, as various optimization tasks are triggered for execution, system throughput increases accordingly. However, this is accompanied by more severe performance jitter. Evidently, different optimization objectives across the different tasks clash, introducing additional noise to the system.

Table 1: Characteristics of state-of-the-art optimization techniques for HSS performance enhancement.

Techniques	Effect	Location	Scope
Orthus CAS Cache (OCach) [44]	Cache traffic optimization	Cache tier	Single-tier
Sibyl [37]	Data placement optimization	Storage tier	Multi-tier
ScaleLFS [14]	In-FS GC optimization	Storage tier	Single-tier
L-QoCo [50]	Cache BW control strategy	Cache tier	Single-tier
LBICA [2]	Cache BW control strategy	Cache tier	Single-tier
PASCAL (our proposed framework)	Collaborative BW orchestration	Entire hierarchy	Multi-tier

Table 2: Direct Data Access (DDA) improves system performance (throughput and latency) but exacerbates jitter for an AI workload (ResNet-50) on HSS. We observed consistent results across different AI workloads. Due to space limitations, we present only the most representative case here and discuss more in the evaluation.

Metrics	Mixture	Mixture+DDA	Mixture+DDA+L-QoCo	Mixture+L-QoCo
Throughput (MB/s)	6912.9	9233.6	10950.3	7622.8
Avg. Latency (ms)	38.7	18.9	13.1	26.8
Throughput Jitter	18.9%	42.1%	31.6%	26.8%

Observation #2. Conflicting optimization objectives across different tasks can significantly disrupt HSS performance.

Inspired by research in the fields of caching and networking that emphasizes the critical impact of bandwidth control on overall system performance [2, 4, 26, 29, 31, 42, 50], we analyze the impact of bandwidth control strategies on HSS performance. We conduct a similar step-by-step experiment as before, but after applying cache optimization (Ocach), we deploy the state-of-the-art caching bandwidth control solution L-QoCo [50]. The results are shown in Fig. 3(c). Clearly, the bandwidth control strategy delivers significant results, substantially reducing performance fluctuations without compromising throughput. However, when other data migration tasks, including tiering and garbage collection, run concurrently, the system still exhibits noticeable performance jitter, eliminating the jitter dampening of L-QoCo altogether. We posit that the reason behind the jitter lies in the fact that L-QoCo is only effective at managing bandwidth for its target object (i.e., the cache), yet it performs poorly in managing bandwidth for multiple data migration tasks within HSS. Due to space constraints and the fact that LBICA [2] demonstrates similar performance to L-QoCo, we omit its results here.

Observation #3. Bandwidth control strategies can effectively reduce performance jitter. However, a single bandwidth control strategy applied solely to a specific task cannot handle the inherent concurrent multitasking in HSS.

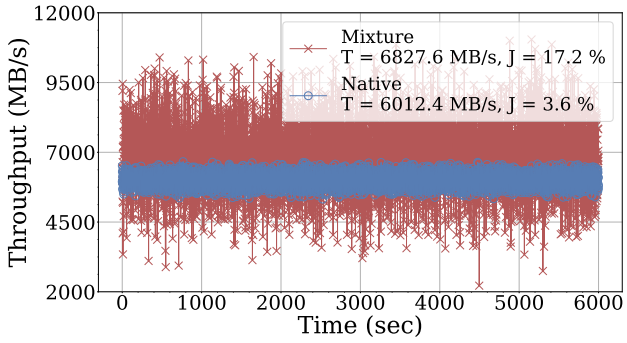
2.2 Observations on Direct Data Access Architectures in HSSs

We quantify the performance impact of emerging Direct Data Access (DDA) architectures (e.g., NPU/GPU Direct) that perform data operations from the HSS directly to accelerators. We conduct an experiment on our commercial HSS using an AI workload (ResNet-50; see details in Section 4). We compare four configurations: the baseline (Mixture), the baseline with NPU Direct (Mixture+DDA), NPU Direct combined with a state-of-the-art bandwidth control strategy, L-QoCo (Mixture+DDA+L-QoCo), and the baseline with L-QoCo alone (Mixture+L-QoCo). The results presented in Table 2 reveal a critical performance trade-off in this system.

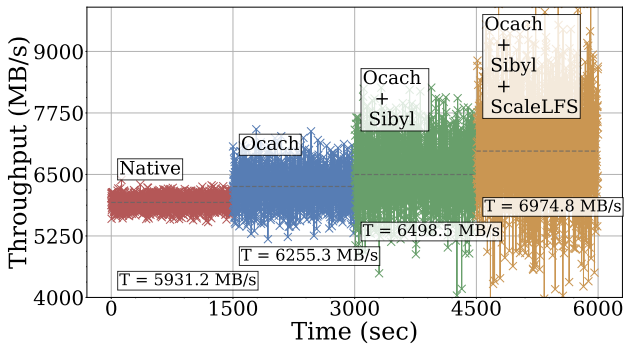
AI training pipelines, characterized by bursty, phased I/O patterns (e.g., alternating epochs of data loading, checkpointing, and computation), inherently create fluctuating and concurrent demands on the storage hierarchy. NPU Direct (Mixture+DDA) amplifies this effect: while it effectively reduces access latency and boosts throughput by removing CPU overhead, it also removes a natural system-level arbiter for concurrent I/O streams. This leads to unregulated, direct contention for shared storage bandwidth among multiple AI tasks or between AI and co-located database workloads, causing performance jitter to surge from 18.9% to 42.1%. The addition of a local bandwidth control strategy like L-QoCo (Mixture+DDA+L-QoCo) partially mitigates the issue but fails to restore stability, as jitter remains 67% above baseline levels. This is because L-QoCo is designed for cache-tier regulation and lacks the global view of pressure propagation across the entire NVRAM-SSD-HDD hierarchy under the direct, high-speed data injection of NPU Direct.

Consequently, the high throughput and low latency promised by direct-access architectures for AI come at the cost of predictability, a trade-off unacceptable for performance-sensitive, mixed-workload deployments. AI workloads not only stress HSS performance but, when combined with NPU Direct architecture, fundamentally transform the bandwidth coordination problem from a local optimization to a system-wide pressure balancing challenge.

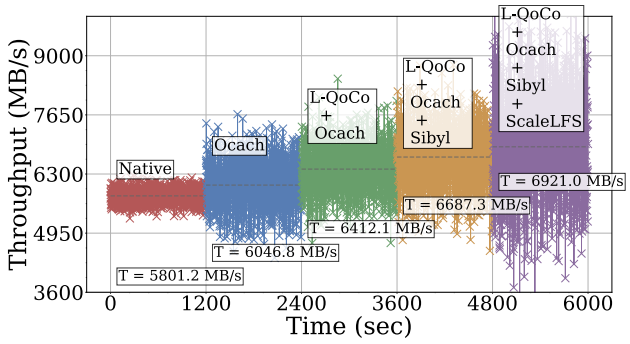
Observation #4. Direct Data Access architectures (NPU/GPU Direct) in HSSs introduce severe performance jitter that existing local bandwidth controllers cannot fully resolve, highlighting the need for system-wide coordination.



(a) Native HSS vs Mixture HSS



(b) Step-by-step experiments



(c) Step-by-step experiments with cache bandwidth control

Figure 3: Impact of concurrent migration tasks on HSS throughput over time (YCSB, configuration T1). (a) Native vs. Mixture: adding state-of-the-art optimizers boosts throughput but increases jitter. (b) Step-by-step addition of each optimizer shows jitter grows with the number of concurrent tasks. (c) Adding cache bandwidth control (L-QoCo) reduces jitter initially, but the benefit vanishes once additional migration tasks (tiering, GC) run concurrently.

2.3 Motivation

The preceding observations reveal a common theme: existing migration optimizers treat tasks in isolation, and single-task bandwidth controllers (e.g., L-QoCo, LBICA [2]) cannot tame the jitter produced by concurrent, interacting flows. This problem is further amplified by Direct Data Access architectures that bypass the CPU and inject high-speed I/O directly into the storage hierarchy (Observation #4). What is needed is a *collaborative* bandwidth orchestration mechanism that (i) establishes a system-wide metric linking tier utilization to migration urgency, and (ii) coordinates bandwidth allocation across all tasks to jointly optimize throughput, jitter, and tail latency.

Drawing an analogy to network congestion, where backpressure-based scheduling propagates queue state across routers to reduce delay and jitter [28, 48], we propose PASCAL, a unified bandwidth orchestration framework inspired by Pascal's law. PASCAL models an HSS as a closed-loop hydraulic network of pressurized tiers, dynamically allocating bandwidth among all concurrent data migration tasks, including I/Os generated by DDA architectures. By doing so, PASCAL aims to unlock the raw performance benefits of direct-access hardware while ensuring throughput stability, low tail latency, and predictable performance demanded by robust, mixed-workload storage systems.

3 PASCAL

PASCAL frames the multi-task bandwidth assignment problem in HSSs through a hydraulic system analogy drawn from *Pascal*'s law: a change in pressure applied to a confined fluid is transmitted undiminished throughout the fluid and its container. In this model, each storage tier is a pressurized vessel, the stored data is the fluid, I/O intensity acts as a heat source that raises pressure, and inter-tier bandwidths are valves (Fig. 2, right). By translating tier utilization into dynamic pressure metrics and coordinating migrations through system-wide pressure gradients, PASCAL achieves adaptive bandwidth control that jointly balances throughput, tail latency, and throughput stability.

Overview. PASCAL splits bandwidth orchestration into two paths that operate on different timescales: a *fast* allocation path that runs every control interval and decides bandwidth from the current state of the hierarchy, and a *slow* tuning path that updates the pressure parameters when the workload or hardware operating point shifts.

The fast path proceeds in three steps. First, the current utilization of each tier is mapped to a scalar *tier pressure*. Second, tier pressures are combined into per-task pressures for both inter-tier migrations and intra-tier maintenance tasks. Third, the available migration bandwidth budget is split across active tasks in proportion to their positive task pressures, subject to physical bandwidth caps and safety reservations. A lightweight efficiency predictor then corrects the commanded rates when the delivered bandwidth diverges from the request.

The slow path uses Bayesian optimization to tune the per-tier pressure parameters that define each tier's target operating point. This lets PASCAL adapt to different hardware configurations and workload phases without complicating the per-interval decision.

Problem formulation: Consider an HSS with N storage tiers and K concurrent data migration tasks (e.g., tiering, garbage collection, cache flush). At each control interval T, PASCAL must decide a bandwidth allocation vector $\mathbf{b} = (b_1, b_2, \ldots, b_K)$ subject to the

aggregate bandwidth budget B of the system. The optimization objective is:

$$\max_{\mathbf{b}} \ \lambda_1 \cdot \text{Throughput}(\mathbf{b}) \ - \ \lambda_2 \cdot \text{Jitter}(\mathbf{b}) \ - \ \lambda_3 \cdot \text{TailLatency}(\mathbf{b}), \quad (1)$$

subject to $\sum_{k=1}^{K} b_k \le B, \quad b_k \ge 0 \ \ \forall k$, and per-tier physical constraints (e.g., PCIe lane bandwidth, DRAM bus bandwidth). Here, Throughput is the client-observed aggregate I/O rate, Jitter is the coefficient of variation of per-interval throughput (defined formally in Equation (7)), and TailLatency is the 99.9th-percentile request latency. The weights $\lambda_1, \lambda_2, \lambda_3 > 0$ encode operator priorities and are absorbed into the Bayesian optimization objective (Section 3).

This optimization is challenging for three reasons. First, the objective functions are not available in closed form, as they depend on workload dynamics, device firmware behavior, and cross-task interference that are difficult to model analytically. Second, the search space grows with the number of tiers and tasks. Third, each evaluation requires running the actual system for a full control interval, making gradient-based methods impractical. PASCAL addresses these challenges through a two-level design: an *inner loop* that computes bandwidth allocations analytically from pressure gradients (fast, closed-form), and an *outer loop* that tunes the pressure parameters via Bayesian optimization (sample-efficient, gradient-free).

Pressure as a unified metric for tier utilization: Pressure is a dynamic, tier-specific metric that quantifies the urgency of data migration. Unlike static "watermark" thresholds, which trigger reactive throttling at fixed capacity levels, pressure reflects real-time deviations from learned optimal watermarks.

Each tier's optimal watermark w_i^* represents the utilization level that maximizes client throughput without risking congestion, which relies on many system factors and parameters. For example, an NVRAM tier might stabilize for a given client access pattern at $w_{NVRAM}^* = 85\%$, while an SSD tier operates optimally at $w_{SSD}^* = 75\%$ for this workload. These targets are not fixed. PASCAL employs Bayesian optimization to iteratively refine w_i^* based on workload patterns and system constraints. During a training phase, PASCAL tests utilization levels, observes their impact on throughput and latency, and models the performance to identify where client I/O is maximized and data migrations are sustainable.

Keeping with the theme of hydraulics and inspired by Poisson's adiabatic process equation, the pressure of each tier p_i is computed as

$$p_i = p_i^* \left(\frac{\max\{0, w_i - w_i^*\}}{\overline{w}_i - w_i} \right)^{\alpha_i}, \quad \text{for } w_i < \overline{w}_i, \quad (2)$$

Here, w_i is the current capacity utilization of tier i, w_i^* is its target watermark (learned, $w_i^* \in [0.2, 0.9]$, default setting 50%), $\overline{w}_i$ is the *maximum safe utilization* of tier i (with $w_i^* < \overline{w}_i \le 1$), p_i^* is the pressure scale factor (learned, $p_i^* \in [0.1, 0.9]$, default setting 0.5), and α_i is the sensitivity exponent (learned, $\alpha_i \in [1.0, 5.0]$, default setting 2.0).

Note that $\overline{w}_i$ is typically set 2–5% below the physical capacity to leave headroom for firmware reserves, GC working space, and emergency burst absorption, and can differ across NVRAM, SSDs, and HDDs depending on write amplification and latency-sensitivity profiles. A high pressure ($w_i \to \overline{w}_i$) signals urgent offloading (e.g., NVRAM near its safe limit), while zero pressure ($w_i \le w_i^*$) indicates that the tier is at or below its target utilization and does not require offloading. When such a tier is below target, it can serve as a low-pressure destination for data migrating from higher-pressure tiers.

The equation relies on pressure differences to coordinate tasks across tiers, following the same principle as backpressure/Max-Weight control: respond to relative congestion to allocate service [39]. The particular pressure nonlinearity we use is a smooth, monotone proxy for backlog; other monotone, convex mappings would yield the same qualitative behavior while trading responsiveness vs. smoothness. This is consistent with the established theory behind differential-queue scheduling.

Design alternatives and rationale: Several design choices in PASCAL merit justification. *Why a nonlinear pressure function?* A simpler linear mapping $p_i \propto (w_i - w_i^*)$ would under-react near full capacity: at 95% utilization the system is critically close to stalling, yet a linear function treats it as only marginally more urgent than 90%. The Poisson-adiabatic form in Equation (2) produces a steep, convex gradient near $\overline{w}_i$, ensuring that the bandwidth allocator responds aggressively before saturation occurs—mirroring the physics of real hydraulic systems where pressure rises sharply as a vessel approaches capacity. *Why pressure-proportional fair-share rather than weighted max-min or PID control?* Max-min fairness aims for equality across flows, but migration tasks are inherently asymmetric: a full NVRAM flush is far more urgent than background GC on a lightly loaded HDD. Proportional allocation lets urgency (pressure) dictate the share directly. PID controllers, on the other hand, require per-task tuning of three gain parameters (K_p, K_i, K_d), and deploying independent PID loops per task reintroduces the very oscillation problem we seek to avoid (as demonstrated empirically in Section 2). *Why Bayesian optimization rather than reinforcement learning or grid search?* The pressure parameters (w_i^*, p_i^*, α_i) define a modest search space (9 dimensions for a three-tier HSS) that must be explored on real hardware where each trial is expensive (minutes to tens of minutes). Bayesian optimization is specifically designed for such low-dimensional, expensive black-box problems [35]. Reinforcement learning, while powerful, typically requires orders of magnitude more samples to converge and would impose unacceptable overhead in a production setting. Grid search scales exponentially with dimensionality and cannot adapt its sampling strategy.

Collaborative migration planning and bandwidth allocation: PASCAL orchestrates data flows by translating pressure gradients into bandwidth allocations, ensuring coordinated action across tiers. For each data migration path, e.g., from NVRAM to SSD, PASCAL starts by computing the pressure difference

$$\Delta p_{ij} = p_i - p_j, \quad (3)$$

where p_i is the source tier's pressure and p_j is the destination tier's pressure. A positive Δp_{ij} indicates migration urgency from tier i to tier j.

The tasks defined in Section 2 may span multiple tiers. We leverage the pressure metric to compute for each *inter-tier* task k a per-task pressure difference $\overline{\Delta p_k}$ by summing up the weighted average of the pressure differences Δp_{ij} across the tiers that a task utilizes ($i, j \in k$, where k denotes a specific task). The weights are the fractions of the task's total data volume transported between

each of the tiers.

$$\overline{\Delta p_k} = \sum_{e_{ij} \in k} \Delta p_{ij} \cdot \frac{\text{task data volume along } e_{ij}}{\text{total task volume}}, \tag{4}$$

where e_{ij} represents the connection between tier i and j.

Intra-tier maintenance tasks. Some tasks, in particular file-system garbage collection on tier i, do not move data across tiers and therefore do not have an ordinary inter-tier pressure gradient. PASCAL models these as *pressure-relief* operations on the affected tier: each intra-tier task is treated as a virtual edge from the tier to a free-space sink whose pressure is zero. The per-task pressure is then proportional to the source tier's pressure, weighted by the operation's reclaim efficiency:

$$\overline{\Delta p_k} = p_i \cdot \rho_k, \qquad \rho_k \in [0.1, 0.9], \tag{5}$$

where ρ_k is the expected free space reclaimed per byte moved by task k (e.g., the invalid-data ratio of the segments selected by GC). This integrates intra-tier maintenance into the same bandwidth allocator as inter-tier flows: when the tier is hot or reclaim is highly effective, GC bandwidth rises; conversely, when the tier is below its target watermark or reclaim is inefficient, GC is naturally deprioritized. A minimum maintenance bandwidth, set by deployment policy, prevents GC starvation when the orchestrator would otherwise zero its allocation while utilization is still close to $\overline{w}_i$.

Client I/O is modeled as an external pressure source that continuously injects data into the HSS, maintaining a non-zero inward pressure gradient. Given these per-task pressure differences, PASCAL dynamically allocates bandwidth using a pressure-proportional fair-share strategy that balances pressures across tiers while respecting real-world execution constraints. In a system with a total available aggregate bandwidth of B, the per task bandwidth b_k is allocated as follows:

$$b_k = B \cdot \frac{\max(0, \overline{\Delta p_k})}{\sum_{\text{all tasks j}} \max(0, \overline{\Delta p_j})}, \tag{6}$$

where high-pressure tasks receive a larger b_k, and low-pressure tasks, such as GC operations during low utilization, are deprioritized.

PASCAL performs its operations on system-level tasks that aggregate all client requests. Thus, PASCAL preserves client QoS and priorities via a standard scheduler.

Efficiency prediction: An additional challenge posed in a real system is that the commanded bandwidth b_k and the actual executed bandwidth often diverge due to system overheads. These overheads stem from processes such as internal SSD GC, storage server CPU load, bus contention, or firmware-level throttling. Ignoring this gap would cause PASCAL to systematically under-provision bandwidth for urgent tasks, defeating the purpose of pressure-aware allocation.

To bridge this gap, PASCAL maintains a per-task *efficiency ratio* $\eta_k(t) = b_k^{\text{actual}}(t) \,/\, b_k^{\text{cmd}}(t)$ at each interval t, and predicts the efficiency for the next interval using three conventional functions applied to a sliding history window of length W (default $W = 10$ intervals):

- **Exponential Moving Average (EMA):** $\hat{\eta}_k^{\text{EMA}}(t) = \beta \cdot \eta_k(t) + (1-\beta) \cdot \hat{\eta}_k^{\text{EMA}}(t-1)$, where $\beta \in (0, 1]$ is a smoothing factor (we use $\beta = 0.3$). EMA reacts quickly to recent changes while dampening noise.

Table 3: Operational settings of PASCAL.

Setting	Value / policy
Control interval T	5 seconds
Default parameters	$w_i^* = 50\%$, $p_i^* = 0.5$, $\alpha_i = 2.0$
Per-tier parameter ranges	$w_i^* \in [20\%, 90\%]$, $p_i^* \in [0.1, 0.9]$, $\alpha_i \in [1.0, 5.0]$
Initial / warm-up samples	5 trials
BO trials per tuning phase	20 trials
Acquisition function	Expected Improvement (EI)
Retuning trigger	Period (600s) or Throughput Drift > 20%
Convergence criterion	Max Trials (20) or EI < 10^{-3}
Fast-path overhead	< 1% CPU
Fallback when BO invalid	Default Native configuration

- **Mean:** $\hat{\eta}_k^{\text{mean}}(t) = \frac{1}{W} \sum_{\tau=t-W+1}^{t} \eta_k(\tau)$. The window average provides a stable estimate under slowly varying conditions.
- **Minimum:** $\hat{\eta}_k^{\min}(t) = \min_{\tau \in [t-W+1,\, t]} \eta_k(\tau)$. The conservative predictor guards against worst-case efficiency drops.

PASCAL selects the predictor that minimizes the absolute prediction error over the most recent window, breaking ties in favor of the more conservative (min) estimate. Given the predicted efficiency $\hat{\eta}_k$, the adjusted commanded bandwidth becomes $b_k^{\text{adj}} = b_k \,/\, \hat{\eta}_k$, so that the *actual* delivered bandwidth approximates the intended allocation b_k. The adjusted value is capped at the per-task physical maximum to prevent over-provisioning.

System-wide coordination and learning: A key property of PASCAL is its ability to propagate pressure changes globally, ensuring tiers collaborate rather than compete.

At every control interval T, PASCAL samples three signals from the HSS: the mean throughput, the throughput jitter defined in Equation (7), and the p99.9 tail latency. It then performs surrogate modeling by using a Gaussian process to approximate the performance as a function of the hyperparameters: per-tier optimal watermark and corresponding pressure values w_i^* and p_i^*, and sensitivity exponent α_i. Using Bayesian optimization, PASCAL performs multi-parameter optimization and selects the values that maximize the composite objective in Equation (1). After calculating the per-task bandwidth, it estimates the expected bandwidth of the actual system and adjusts the per-task bandwidth provisioning accordingly. PASCAL then continues this cycle to ensure system stability while maximizing the system's outward throughput. Bayesian optimization is sample-efficient and is intended for expensive evaluations. The computational overhead is negligible for the small trial counts we use [35].

PASCAL separates a fast path that allocates bandwidth every control interval from a slow path that retunes the pressure parameters via Bayesian optimization. The fast path is closed-form (Equation (2)–Equation (4)) and runs deterministically on every interval. The slow path proposes a new (w_i^*, p_i^*, α_i) vector after observing throughput, p99.9 latency, and throughput jitter over a window of completed intervals. It is sample-efficient because each evaluation requires running the actual system. PASCAL uses a small number of warm-up samples to bootstrap the Gaussian process and then iterates a fixed number of BO trials per tuning phase. Table 3 summarizes the operational settings used in our deployment.

PASCAL caps the per-task bandwidth with respect to shared limits like PCIe/DRAM, and then splits it by pressure across tiers. This mirrors standard Network Utility Maximization thinking: a global capacity with local prices or pressures yields stable, fair allocations.

Example workflow: To illustrate PASCAL's coordination, consider a real-world workload burst during an HSS file system garbage collection task (FS GC):

(1) NVRAM utilization increases ($w_{NVRAM} > w^*_{NVRAM}$).
(2) This increases $\Delta p_{NVRAM,SSD}$ and triggers an NVRAM to SSD migration, with a respective increase in bandwidth allocation to the SSDs.
(3) PASCAL recomputes the pressure gradients of the SSDs, reducing the bandwidth allocated to FS GC on the SSDs, thereby boosting the migration bandwidth.
(4) The respective pressure on the SSD tier increases due to the incoming data stream, therefore a higher bandwidth to the data migration task between the SSD to HDD is allocated.
(5) The NVRAM utilization drops, and the client I/O jitter stabilizes.

4 EVALUATION

We evaluate PASCAL to determine whether its hydraulic-inspired coordination improves HSS performance over current state-of-the-art methods.

Testbed: We evaluate on a commercial off-the-shelf hierarchical storage system (Huawei OceanStor Dorado 18000 V6 [21]) under multiple configurations.

①**T1** is equipped with 4 × 48-core 2.6 GHz Kunpeng-920 ARM-based CPUs [20], 16 × 32 GB DDR4-2666 (serving as the NVRAM tier), 34 × 0.96 TB SAS SSDs (SSD tier), 96 × 0.96 TB SAS HDDs (HDD tier), and Ascend 910B NPUs [19] for Direct Data Access experiments. This configuration reflects a typical three-tier enterprise HSS with high-speed memory, a flash tier for capacity and performance, and a large-capacity disk tier.

②**T2** halves the flash devices to 17 SSDs based on **T1**.

③**T3** removes the HDD tier based on **T1**, yielding a two-tier configuration.

The three configurations stress different migration-interference regimes: **T1** is the full three-tier baseline; **T2** halves the SSD tier, amplifying SSD-side contention between tiering, cache flush, and maintenance; **T3** removes the HDD tier, tightening the cache-flush and SSD-GC coupling with no HDD "shock absorber" downstream. Together they cover distinct capacity-to-bandwidth ratios and tier topologies, so improvements cannot be attributed to a single hardware operating point.

PASCAL is integrated into the OceanStor controller software stack and has been deployed for customer use across the OceanStor product series. Our evaluation runs on the production-grade hardware testbeds (**T1**–**T3**) described above within a large-scale Huawei datacenter, with workloads that are either customer-derived traces or established benchmarks.

Baselines: We compare PASCAL against the following configurations:

①**Native:** The native vendor default HSS configuration, using the vendor's standard mechanisms and policies [18]. Concretely, each tier uses LRU for replacement; garbage collection employs the mark–compact algorithm [15]; and SmartTier [22] provides tiering.

②**Mixture (Mix.):** A composite configuration that augments Native with three state-of-the-art optimizations: Ocach [44] for cache traffic, Sibyl [37] for tiering, and ScaleLFS [14] for in-file-system GC (see Table 1). In the evaluation tables, "+Mix." denotes that the respective bandwidth controller operates atop this configuration.

③**LBICA:** A rule-based bandwidth control method that mitigates I/O-cache bottlenecks during bursts via adaptive write-policy management [2].

④**L-QoCo:** An online reinforcement-learning cache-bandwidth control technique [50].

These baselines cover the vendor default, a composite state-of-the-art configuration, rule-based, and learning-based bandwidth control strategies, enabling a balanced evaluation across design choices.

Workloads: We evaluate all strategies using diverse workloads (YCSB and ZippyDB in database, MLPerf-Storage in AI/LLM):

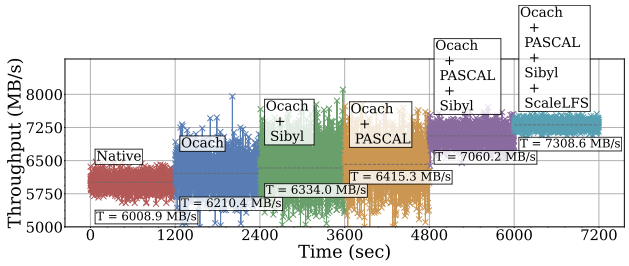
- **YCSB:** Standard YCSB-based storage workloads (W1) [8], to stress key-value-style access patterns.
- **ZippyDB:** The Facebook ZippyDB benchmark (W2) [6], to capture production-like RocksDB behavior.
- **MLPerf-Storage:** I/O traces collected from MLPerf-Storage test suite for the U-Net3D (W3), ResNet 50 (W4) and CosmoFlow (W5) workloads [1].
- **Real-world workloads:** Three anonymized I/O traces collected from three data centers. Workload W6-a originates from one of China's largest banks, supporting its core database operations and data analytics services. Workload W6-b is derived from a leading AI company specializing in large language model (LLM) training and workload W6-c is sourced from one of Asia's most extensive AI inference service platforms. For each category we consolidate traces and construct a single 6,000-second workload to reflect sustained production conditions while preserving temporal dynamics.

Together, workloads W1-W6 cover controlled microbenchmarks and production-representative access patterns, allowing us to quantify both peak and steady-state behavior under diverse conditions.

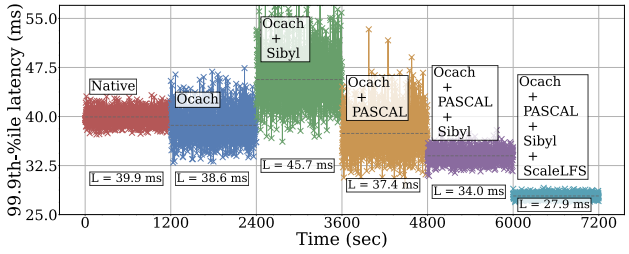
Metrics: We report client-observed 99.9th-percentile (p99.9) latency, mean throughput, and throughput jitter. Latency is computed over per-request completion times; throughput is the aggregate rate measured at the client boundary. Throughput jitter quantifies short-term variability in service rate and is defined as the (dimensionless) coefficient of variation of per-interval throughput:

$$\sqrt{\frac{\sum_{t=0}^{T}(I_t - \overline{I_t})^2}{T}} / \overline{I_t}, \tag{7}$$

where I_t is the throughput at time interval t and $\overline{I_t}$ is the mean throughput. A well-tuned HSS should deliver high throughput with low tail latency and low jitter.



(a) Throughput



(b) P99.9 latency

Figure 4: ZippyDB (W2) throughput and tail latency over time under different configurations on a commercial HSS. PASCAL stabilizes throughput and reduces tail spikes as more migration tasks are coordinated.

4.1 Effectiveness of PASCAL

This subsection evaluates whether PASCAL effectively optimizes concurrent data migrations and, in turn, improves end-to-end service for foreground I/O. Unless otherwise stated, the time-series study in this subsection uses the **T1** testbed described above. We use the ZippyDB workload (W2) and report time-series throughput (Fig. 4(a)) and p99.9 latency (Fig. 4(b)). We omit other workload results for brevity, as they demonstrate similar performance trends in this scenario.

Under the Native configuration, the system sustains an average throughput of 6008 MB/s with p99.9 latency of 39.9 ms. As visible in the figures, short-term fluctuations in both throughput and latency are present but not dominant; the platform remains generally stable, providing a reasonable baseline against which to compare migration-aware techniques.

When we enable Ocach [44] on top of Native, mean throughput increases by 3.3%, indicating that smarter migration decisions can free useful I/O capacity for the foreground workload. However, the figures also show more pronounced variability: throughput jitter rises and p99.9 latency exhibits larger spikes. Intuitively, a single migration optimizer can intensify burstiness by opportunistically consuming bandwidth without regard to the instantaneous state of other background activities or the client-facing queue, leading to transient queue buildup and tail inflation.

Applying PASCAL in this scenario, i.e., PASCAL operating in a setting with only *one* migration task, yields a modest increase in throughput and a moderate reduction in p99.9 latency relative to Native. Jitter, however, changes little. This behavior is expected: PASCAL's hydraulic-inspired coordination is designed to arbitrate *across* concurrent flows. With only one migration flow, there is limited scope for cross-flow coordination, so the principal gains come from gentle rate shaping and avoiding interference with the foreground path rather than from smoothing interactions among multiple migrations.

We then evaluate Sibyl [37] alongside Ocach, with and without PASCAL. The system now runs two distinct migration tasks whose bandwidth demands can interfere without proper coordination. In this setting, PASCAL delivers three effects visible in the figures: (i) a sharp increase in sustained throughput, (ii) a marked drop in p99.9 latency, and most importantly, (iii) a substantial reduction in throughput jitter and tail-latency spikes. These improvements arise because PASCAL continuously balances the "pressure" of concurrent migration flows against the foreground service rate, throttling, deferring, or accelerating individual migrations to prevent queue spikes and head-of-line blocking. As a result, foreground I/O experiences fewer bursts and shorter tails.

When enabling additional migration tasks, such as ScaleLFS [14], PASCAL identifies additional opportunities to reallocate bandwidth among these background tasks, further smoothing their aggregate demand and protecting the client-visible path. The figure shows the same pattern as with Sibyl, often even more pronounced: improved throughput, lower p99.9 latency, and noticeably reduced jitter, reflecting PASCAL's ability to orchestrate multiple, heterogeneous migration flows.

In summary, PASCAL's benefits grow with the number of concurrent migration tasks: a single task sees modest shaping gains, while two or more tasks unlock substantial interference reduction, lower jitter, and improved throughput and tail latency. This validates the core premise that cross-flow coordination is most impactful when the system must juggle multiple competing background migrations.

4.2 Comprehensive comparison

Aggregate results. We complement the time-series study with the aggregate view in Table 4, which reports client-observed mean throughput, p99.9 tail latency ("Tail latency"), and throughput jitter. LBICA and L-QoCo were originally designed as local controllers. To the best of our knowledge, there is no existing "out-of-the-box" global orchestration framework specifically for multi-task HSS bandwidth management. We therefore construct a strong composed industrial-practice baseline by deploying these local controllers alongside advanced migration optimizers (Ocach, Sibyl, and ScaleLFS). "LBICA + Mix." and "L-QoCo + Mix." refer to this composed configuration. "PASCAL + Mix." replaces the per-task local controllers with a single global pressure-driven orchestrator. Our improvements should therefore be read as gains over local-only policies and their composed global application.

Table 4 shows that applying LBICA and L-QoCo atop these state-of-the-art policies generally trims tails and nudges throughput upward, but jitter remains high. These controllers still react locally and do not arbitrate across migration tasks, so interference is only partially mitigated.

With PASCAL orchestrating migration bandwidths, the system achieves the strongest combined improvements across W1–W5 under **T1**: mean throughput improves by 14–27% and p99.9 tails drop by 34–72% versus Native. Importantly, PASCAL suppresses the

Table 4: Summary of performance evaluation results for the workloads W1–W5 under different methods and system configurations (T1–T3). Although the system default configuration (Native) achieves lower jitter, PASCAL attains similar jitter but much higher throughput and lower latency.

Workload & Method		T1			T2			T3		
Workload	Method	Mean throu. (MB/s)	Tail latency (ms)	Throu. jitter	Mean throu. (MB/s)	Tail latency (ms)	Throu. jitter	Mean throu. (MB/s)	Tail latency (ms)	Throu. jitter
W1 **YCSB**	Native	5025.3	45.9	**3.1%**	3976.2	53.7	7.5%	3149.6	31.5	5.3%
	LBICA + Mix.	5421.9	34.8	30.7%	4761.3	39.7	19.8%	3795.8	16.9	14.3%
	L-QoCo + Mix.	5488.5	**30.2**	27.2%	4680.1	28.6	19.2%	3922.3	11.6	10.9%
	PASCAL + Mix.	**5739.2**	30.3	6.3%	**5341.0**	**15.1**	**7.0%**	**4199.2**	**7.5**	**3.3%**
W2 **ZippyDB**	Native	6612.8	29.5	12.2%	5520.2	43.2	9.1%	4795.3	24.1	**7.2%**
	LBICA + Mix.	6681.2	19.1	23.5%	6197.1	26.7	18.6%	5302.1	15.2	18.6%
	L-QoCo + Mix.	6782.9	**15.0**	20.4%	6315.7	26.6	15.4%	5527.2	13.6	15.4%
	PASCAL + Mix.	**7510.1**	16.6	**9.5%**	**6903.4**	**22.5**	**7.2%**	**6033.8**	**10.1**	**7.2%**
W3 **U-Net3D**	Native	7253.8	41.5	13.8%	6177.0	58.1	10.6%	5708.8	33.9	10.6%
	LBICA + Mix.	7890.2	31.2	29.8%	6751.7	36.9	31.1%	6429.3	18.3	31.1%
	L-QoCo + Mix.	7997.5	34.9	22.5%	7109.6	31.3	25.9%	6615.2	16.4	25.9%
	PASCAL + Mix.	**8601.5**	**11.5**	**11.3%**	**7885.9**	**22.0**	**8.6%**	**7236.1**	**12.2**	**8.6%**
W4 **ResNet 50**	Native	5753.7	35.1	9.4%	4822.6	49.2	17.4%	4391.5	26.8	**6.2%**
	LBICA + Mix.	6581.3	27.8	16.1%	5369.9	37.4	27.3%	5033.4	16.2	21.3%
	L-QoCo + Mix.	6741.0	19.4	16.7%	5780.2	27.5	22.7%	5268.5	15.5	15.3%
	PASCAL + Mix.	**7332.8**	**10.6**	**7.2%**	**6616.5**	**14.3**	**12.3%**	**5805.7**	**8.9**	10.2%
W5 **CosmoFlow**	Native	6972.1	25.2	13.9%	5588.9	36.9	10.1%	5128.9	21.0	9.2%
	LBICA + Mix.	7309.5	13.6	19.7%	6423.7	22.7	21.4%	5877.5	17.3	22.9%
	L-QoCo + Mix.	7961.2	10.2	13.8%	6531.5	18.3	16.9%	5922.2	16.6	16.9%
	PASCAL + Mix.	**8311.5**	**7.2**	**8.3%**	**7238.1**	**12.5**	**7.6%**	**6428.6**	**10.5**	**6.6%**

volatility introduced by the application of the mixture of optimizations; relative to LBICA and L-QoCo, PASCAL cuts their throughput jitter by 40–79% while maintaining high throughput and low latency levels. Specifically for W3–W5, PASCAL achieves up to 13% higher throughput and 67% lower tail latency. This demonstrates PASCAL's strong performance on machine learning workloads. Overall, the cross-workload results substantiate the central claim: coordination, not just local optimization, is key to stable, high-performance tiered storage under concurrent data migrations.

T2 and **T3** show the same qualitative trends under different system constraints:

Under **T2**, which halves the SSD tier, PASCAL improves throughput by 25–53% and reduces tails by 48–72% vs. Native across W1–W5, while keeping jitter at or below Native on nearly all workloads. Compared to LBICA and L-QoCo, PASCAL raises throughput by 7–20%, lowers tails substantially by 15–62%, and decreases throughput jitter by 53–72%. The overall gain exceeds that of T1 because the reduced flash amplifies cross-task contention; PASCAL's global rate control prevents SSD saturation and queue spikes that local policies cannot anticipate.

Under **T3** with the HDD tier removed, compared to LBICA and L-QoCo, PASCAL achieves a 7–15% increase in throughput, a 25–55% reduction in tail latency, and a 25–77% decrease in jitter. The two-tier pipeline tightens the coupling between cache flushes and SSD GC. Without HDDs as the GC shock absorber, local policies can overwhelm the SSDs. PASCAL's pressure-aware orchestration preserves headroom, avoiding bursts while sustaining higher steady-state throughput.

4.3 Ablation study of PASCAL components

We isolate the essential components of PASCAL on W2/T1 by comparing full PASCAL+Mix. against three variants. *Native + fixed global watermarks* replaces pressure-driven allocation with a static per-tier watermark and round-robin bandwidth split, isolating the value of dynamic coordination. *PASCAL-Linear* replaces the non-linear mapping of Equation (2) with a clamped linear pressure

$$P_i = \max\Big(0,\ \min\Big(1,\ \tfrac{v_i - \mathrm{low}_i}{\mathrm{high}_i - \mathrm{low}_i}\Big)\Big), \tag{8}$$

where $\mathrm{low}_i \in [20\%, 40\%]$ and $\mathrm{high}_i \in [80\%, 90\%]$, so $\partial P_i / \partial v_i$ is constant and the system does not raise urgency as the tier approaches capacity. *PASCAL-NoCorr* disables the efficiency predictor ($\eta_k = 1$ for all tasks) so allocation uses raw pressure ratios without correcting for execution bias.

Table 5 reports the results. Compared to fixed watermarks, PASCAL's dynamic coordination improves throughput by 27.9% and reduces tail latency by 51.9%. Full PASCAL also outperforms PASCAL-Linear, showing that the non-linear pressure model provides a steeper, more timely response near capacity. Removing the efficiency predictor nearly doubles throughput jitter (9.8% → 18.7%), confirming that predictive correction is essential when hardware-level bandwidth deviates from commanded values.

4.4 Effectiveness for Direct Data Access in HSSs

To comprehensively evaluate the efficacy of PASCAL in managing the bandwidth contention introduced by direct data access architectures, we conduct experiments using three representative AI workloads from the MLPerf Storage benchmark suite: W3 (U-Net3D), W4 (ResNet-50), and W5 (CosmoFlow). The key results are summarized in Table 6. While DDA delivers substantial raw

Table 5: Ablation study of PASCAL against the simplest global-rule baselines on W2/T1. Evaluating the performance impact of fixed global watermark, non-linear pressure model and efficiency predictor.

Variant	Mean throu. (MB/s)	Tail lat. (ms)	Throu. jitter
Native + fixed global watermarks	5989.3	39.3	28.6%
PASCAL-Linear	7090.6	19.9	16.5%
PASCAL-NoCorr	6903.1	22.3	18.7%
Full PASCAL+Mix.	**7661.9**	**18.9**	**9.8%**

performance gains, it critically exacerbates performance instability. Enabling DDA (Mix.+DDA) consistently improves mean throughput and reduces tail latency across all workloads and tiers compared to the baseline (Mix.). For instance, for W4 on T1, throughput increases by 34% (from 6912.9 to 9233.6 MB/s) and tail latency drops by 51% (from 38.7 to 18.9 ms). However, this comes at a severe cost: throughput jitter increases dramatically, by 123% for W4/T1 (from 21.9% to 42.1%) and by 55% for W3/T1 (from 19.0% to 29.5%). This confirms our earlier observation that DDA, by removing the CPU as a central coordinator, allows uncontrolled bandwidth contention to manifest directly as high performance variability.

PASCAL uniquely harnesses the benefits of DDA while ensuring system-wide stability, outperforming tier-specific coordination. The addition of L-QoCo (Mix.+DDA+L-QoCo) mitigates jitter to some extent but fails to restore it to baseline levels, as it lacks a global view of inter-tier pressure. In contrast, PASCAL consistently achieves the best balance of high performance, low latency, and minimal jitter. For example, for W4/T1, PASCAL boosts throughput to 12319.6 MB/s (a 78% improvement over baseline), reduces tail latency to 12.7 ms, and, most importantly, cuts the severe jitter induced by DDA by nearly two-thirds, down to a stable 15.2%.

These results confirm that PASCAL resolves the throughput-vs-stability trade-off introduced by DDA: its pressure-aware orchestration lets concurrent migrations fully exploit DDA's high-speed pathways while maintaining the predictable performance required by mixed-workload HSS deployments.

4.5 Adaptability for Real-world Workloads

For the real-world workloads, we compare Mix., Mix.+PASCAL, Mix.+DDA, and Mix.+DDA+PASCAL; Native and single-task baselines (LBICA, L-QoCo) are omitted because the production environments already deploy the Mixture optimizations. We evaluate three industry workloads: W6-a (latency-critical OLTP and analytics from a major bank), W6-b (bursty LLM training with periodic checkpointing), and W6-c (sustained high-concurrency AI inference).

Table 7 summarizes the results. Across all three production scenarios, PASCAL resolves the throughput-stability trade-off introduced by DDA. On W6-a/T1, Mix.+DDA+PASCAL reduces the DDA-induced tail latency by 36.7% (26.7 ms → 16.9 ms) while cutting jitter by 56.7% (33.7% → 14.6%) and boosting throughput by 15.4%—essential for transaction SLAs. On bandwidth-hungry W6-b, PASCAL retains DDA's peak throughput (11,358.7 MB/s on T1) while halving its jitter, keeping training progress stable. On steady-stream W6-c, PASCAL+DDA achieves near-optimal latency and reduces jitter by over 50% versus DDA-only across all tiers. The pattern holds across T1–T3: e.g., W6-a/T3 jitter drops from 38.6% to 11.2% with no throughput loss, confirming that pressure-aware orchestration generalizes across I/O patterns.

4.6 Operational Lessons and Failure Analysis

Our deployment experience suggests that the main risk in global migration orchestration is not loss of peak throughput, but unstable reactions to workload phase changes. PASCAL's design therefore keeps the fast path deterministic and bounded, with all online learning pushed to a slow outer loop.

Workload shifts and oscillation risk. Pressure parameters (w_i^*, p_i^*, α_i) are changed only by the BO outer loop. Independent per-task controllers tend to "fight" one another, the failure mode observed in Section 2 for L-QoCo + Mixture; PASCAL avoids this by single-arbiter design (centralized allocation from current pressures, capped per task, updated only at fixed intervals) and an efficiency-prediction window that smooths short-term shocks. We did not observe persistent oscillations in any run.

GC starvation safeguard. Because GC bandwidth is proportional to tier pressure (Equation (5)), it can drop near zero when the tier is below its target watermark. To prevent the tier from drifting toward $\overline{w}_i$, a minimum maintenance bandwidth takes precedence over the pressure-driven allocation.

Robustness and retuning. The optimal parameters (w_i^*, p_i^*, α_i) are sensitive to the underlying throughput-to-latency curves and the workload's burstiness, so parameters learned for a read-heavy trace do not transfer to write-intensive ones. PASCAL therefore retunes BO under two conditions: a periodic refresh every 600 seconds, or a dynamic trigger on a $> 20\%$ drift in throughput patterns like significant workload bursts.

Online overhead. In production, the BO engine runs at a 5-second control interval and converges in roughly 20 trials. The fast-path overhead on the OceanStor controller stays below 1% CPU and has no measurable impact on client I/O.

Deployment lesson. Operators require an interpretable, closed-form fast path: every bandwidth decision must be explainable from a small set of pressure values. Black-box per-task controllers were hard to reason about during incident reviews; we believe this is the main reason a backpressure-style orchestrator is a better industrial fit than end-to-end RL for migration coordination.

5 RELATED WORK

Prior work on data migration in storage systems falls into two camps. "How to" research decides *what* to migrate and via which path: garbage-collection strategies within single-tier storage [13, 25, 34, 38], inter-tier data migration, and cross-tier data placement in multi-tier storage [23, 32, 33, 37, 44, 51]. "How much" research decides the bandwidth allocated to those flows [2, 29, 50].

How to. GC-side techniques include semipreemptible GC [30], host-offloaded segment compaction over ZNS [16], GC-free log-structured filesystems via partition-size decoupling [24], reclaiming invalid sections through underlying flash GC [25], and scalable GC on commodity SSDs [13]. These optimize GC within a single tier; PASCAL complements them by coordinating bandwidth *across* tiers. Cross-tier placement and tiering have been studied via

Table 6: Evaluation with MLPerf workloads under Direct Data Access (DDA). DDA boosts throughput but amplifies jitter on HSS. PASCAL effectively harnesses the performance benefits of DDA while eliminating the severe performance jitter it introduces, thereby unifying high throughput with system stability.

Workload & Method		T1			T2			T3		
Workload	Method	Mean throu. (MB/s)	Tail latency (ms)	Throu. jitter	Mean throu. (MB/s)	Tail latency (ms)	Throu. jitter	Mean throu. (MB/s)	Tail latency (ms)	Throu. jitter
W3 **U-Net3D**	Mix.	4663.1	31.4	19.0%	3224.1	43.6	27.7%	3008.1	32.7	19.6%
	Mix.+DDA	6990.3	18.0	29.5%	5337.8	25.7	38.4%	4483.5	17.9	28.4%
	Mix.+DDA+L-QoCo	7133.6	16.5	22.5%	5902.3	21.5	29.5%	5009.4	14.6	21.3%
	Mix.+DDA+PASCAL	**8122.1**	**13.1**	**10.3%**	**6883.5**	**13.7**	**15.2%**	**6093.1**	**9.8**	**9.8%**
W4 **ResNet 50**	Mix.	6912.9	38.7	21.9%	5324.8	47.3	24.9%	4205.1	39.0	18.04%
	Mix.+DDA	9233.6	18.9	42.1%	8556.1	22.3	38.6%	7802.3	22.6	34.1%
	Mix.+DDA+L-QoCo	10950.3	13.1	31.6%	9103.7	18.7	30.0%	8151.2	19.8	27.9%
	Mix.+DDA+PASCAL	**12319.6**	**12.7**	**15.2%**	**11165.4**	**13.7**	**7.1%**	**9086.2**	**11.6**	**10.7%**
W5 **CosmoFlow**	Mix.	6223.9	48.2	22.0%	4997.2	52.7	19.5%	4227.8	33.9	18.8%
	Mix.+DDA	8762.1	27.3	35.8%	7224.8	39.4	25.3%	6684.0	18.3	27.9%
	Mix.+DDA+L-QoCo	9223.0	23.8	26.9%	7820.1	31.3	24.9%	7557.5	**16.4**	22.5%
	Mix.+DDA+PASCAL	**11009.3**	**18.2**	**13.9%**	**8885.7**	**26.6**	**12.2%**	**8006.2**	16.5	**10.3%**

Table 7: Real-world workload results (W6-a/b/c) under HSS with and without DDA. PASCAL delivers the best throughput–jitter trade-off across all three production workloads, hardware configurations and different architectures.

Real-world Workload & Method		T1			T2			T3		
Workload	Method	Mean throu. (MB/s)	Tail latency (ms)	Throu. jitter	Mean throu. (MB/s)	Tail latency (ms)	Throu. jitter	Mean throu. (MB/s)	Tail latency (ms)	Throu. jitter
W6-a **Bank**	Mix.	5789.2	50.5	24.5%	4762.9	59.2	26.8%	4228.7	38.9	19.7%
	Mix.+PASCAL	**7221.0**	**32.3**	**12.9%**	**5378.5**	**44.5**	**11.7%**	**4890.2**	**24.9**	**15.5%**
	Mix.+DDA	8997.1	26.7	33.7%	7282.8	27.6	43.8%	6753.2	21.4	38.6%
	Mix.+DDA+PASCAL	**10378.9**	**16.9**	**14.6%**	**8179.0**	**20.3**	**14.5%**	**6330.7**	**16.6**	**11.2%**
W6-b **AI Training**	Mix.	6889.2	49.6	25.9%	5444.3	56.3	21.8%	4889.5	39.6	20.7%
	Mix.+PASCAL	**7563.3**	**39.5**	**13.9%**	**5770.8**	**46.2**	**17.2%**	**5441.9**	**30.6**	**13.8%**
	Mix.+DDA	9923.0	26.6	33.7%	8846.0	33.2	29.0%	7531.1	23.8	29.8%
	Mix.+DDA+PASCAL	**11358.7**	**21.9**	**14.6%**	**9456.5**	**26.0**	**14.7%**	**7905.2**	**20.9**	**15.5%**
W6-c **AI Inference**	Mix.	5586.9	38.5	19.9%	5122.4	41.5	21.9%	4763.2	35.8	19.7%
	Mix.+PASCAL	**6031.2**	27.8	**8.2%**	**5782.3**	**37.2**	**11.0%**	**5339.1**	**25.0**	**8.6%**
	Mix.+DDA	8345.1	**21.1**	38.5%	7650.9	24.8	37.2%	6880.5	22.7	28.6%
	Mix.+DDA+PASCAL	**9450.2**	22.9	**18.6%**	**8047.9**	**22.0**	**13.8%**	**7223.1**	**17.3**	**15.2%**

initial-block allocation and migration for hybrid devices [45], extent-based dynamic tiering [11], SSD-as-cache hybrid systems [46], RL-based placement [37, 51], ML-assisted layout [32], NVMe flexible data placement [3], bottleneck-set bandwidth allocation [41], non-hierarchical caching [44], and meta-layer aggregation across medium-specific file systems [33]. These target *what* to migrate; PASCAL operates at a complementary level by governing *how much* bandwidth each concurrent task receives.

How much. LBICA [2] mitigates I/O-cache bottlenecks via adaptive write policies; SIB [26] redirects read I/O from a saturated SSD cache to idle HDDs; L-QoCo [50] applies RL to cache-bandwidth control; and Robus [29] focuses on single-tier cache allocation and bandwidth control. These all target a single task. PASCAL extends this line to the multi-task setting with a pressure-based system-wide metric that collaboratively regulates concurrent migration flows.

6 CONCLUSION

Modern HSSs must coordinate, not isolate, their concurrent migration tasks. Combining state-of-the-art per-task optimizers raises throughput but introduces severe jitter that single-task bandwidth controllers cannot resolve. We propose PASCAL, a pressure-aware bandwidth orchestration framework grounded in Pascal's law: each tier is a pressurized vessel, utilization translates into dynamic pressure, and pressure gradients drive collaborative bandwidth allocation across all migration flows, with Bayesian optimization tuning the operating points online. Across three hardware configurations and eight workloads, PASCAL achieves up to 20% higher throughput, 67% lower tail latency, and 79% reduced throughput jitter over local state-of-the-art controllers and the composed industrial practice, and cuts DDA-induced jitter by up to two-thirds while preserving DDA's throughput and latency benefits.

REFERENCES

[1] 2025. MLPerf Storage Benchmark Suite. https://github.com/mlcommons/storage. [Online; accessed 3-Nov-2025].

[2] Saba Ahmadian, Reza Salkhordeh, and Hossein Asadi. 2019. LBICA: A Load Balancer for I/O Cache Architectures. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1196–1201.

[3] Michael Allison, Arun George, Javier González, and etc. 2025. Towards Efficient Flash Caches with Emerging NVMe Flexible Data Placement SSDs. In *Proceedings of the Twentieth European Conference on Computer Systems, EuroSys 2025,*

Rotterdam, The Netherlands, 30 March 2025 - 3 April 2025. ACM, 1142–1160.

[4] Gaetano F. Anastasi, Massimo Coppola, Patrizio Dazzi, and Marco Distefano. 2016. QoS Guarantees for Network Bandwidth in Private Clouds. *Procedia Computer Science* 97 (2016), 4–13. 2nd International Conference on Cloud Forward: From Distributed to Complete Computing.

[5] Kai Bu, Meng Wang, Hongshan Nie, Wei Huang, and Bo Li. 2012. The Optimization of the Hierarchical Storage System Based on the Hybrid SSD Technology. In *2012 Second International Conference on Intelligent System Design and Engineering Application*. 1323–1326. https://doi.org/10.1109/ISdea.2012.590

[6] Zhichao Cao, Siying Dong, Sagar Vemuri, and David H.C. Du. 2020. Characterizing, Modeling, and Benchmarking RocksDB Key-Value Workloads at Facebook. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*. USENIX Association, Santa Clara, CA, 209–223. https://www.usenix.org/conference/fast20/presentation/cao-zhichao

[7] Hamza Djigal, Linfeng Liu, Jian Luo, and Jia Xu. 2022. BUDA: Budget and Deadline Aware Scheduling Algorithm for Task Graphs in Heterogeneous Systems. In *2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*. 1–10. https://doi.org/10.1109/IWQoS54832.2022.9812865

[8] Subramanya R. Dulloor, Amitabha Roy, and etc. 2016. Data tiering in heterogeneous memory systems. In *Proceedings of the Eleventh European Conference on Computer Systems* (London, United Kingdom) *(EuroSys '16)*. Association for Computing Machinery, New York, NY, USA, Article 15, 16 pages. https://doi.org/10.1145/2901318.2901344

[9] Yicheng Gao and Giuliano Casale. 2022. JCSP: Joint Caching and Service Placement for Edge Computing Systems. In *2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*. 1–10. https://doi.org/10.1109/IWQoS54832.2022.9812888

[10] Pawan Goyal, Dharmendra S. Modha, and Renu Tewari. 2003. CacheCOW: providing QoS for storage system caches. *SIGMETRICS Perform. Eval. Rev.* 31, 1 (jun 2003), 306–307. https://doi.org/10.1145/885651.781070

[11] Jorge Guerra, Himabindu Pucha, Joseph S. Glider, Wendy Belluomini, and Raju Rangaswami. 2011. Cost Effective Storage using Extent Based Dynamic Tiering. In *9th USENIX Conference on File and Storage Technologies, San Jose, CA, USA, February 15-17, 2011*. USENIX, 273–286.

[12] Jing Guo, Zihao Chang, Sa Wang, Haiyang Ding, Yihui Feng, Liang Mao, and Yungang Bao. 2019. Who limits the resource efficiency of my datacenter: an analysis of Alibaba datacenter traces. In *Proceedings of the International Symposium on Quality of Service* (Phoenix, Arizona) *(IWQoS '19)*. Association for Computing Machinery, New York, NY, USA, Article 39, 10 pages.

[13] Jinyong Ha, Sangjin Lee, Hyeonsang Eom, and Yongseok Son. 2025. ScaleLFS: A Log-Structured File System with Scalable Garbage Collection for Commodity SSDs. In *23rd USENIX Conference on File and Storage Technologies, FAST 2025, Santa Clara, CA, February 25-27, 2025*. USENIX Association, 53–67.

[14] Jin Yong Ha, Sangjin Lee, Hyeonsang Eom, and Yongseok Son. 2025. ScaleLFS: A Log-Structured File System with Scalable Garbage Collection for Commodity SSDs. In *23rd USENIX Conference on File and Storage Technologies (FAST 25)*. USENIX Association, Santa Clara, CA, 53–67. https://www.usenix.org/conference/fast25/presentation/ha

[15] B. K. Haddon and W. M. Waite. 1967. A Compaction Procedure for Variable-Length Storage Elements. *Comput. J.* 10, 2 (08 1967), 162–165. https://doi.org/10.1093/comjnl/10.2.162 arXiv:https://academic.oup.com/comjnl/article-pdf/10/2/162/963555/10-2-162.pdf

[16] Kyuhwa Han, Hyunho Gwak, Dongkun Shin, and Jooyoung Hwang. 2021. ZNS+: Advanced Zoned Namespace Interface for Supporting In-Storage Zone Compaction. In *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16, 2021*. USENIX Association, 147–162.

[17] Piao Hu, Yunfei Gu, Ranhao Jia, Chentao Wu, Minyi Guo, and Jie Li. 2022. PRM: An Efficient Partial Recovery Method to Accelerate Training Data Reconstruction for Distributed Deep Learning Applications in Cloud Storage Systems. In *2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*. 1–10. https://doi.org/10.1109/IWQoS54832.2022.9812919

[18] Huawei. 2021. Understanding the Cache Performance of Huawei Oceanstor Storage System V300R003. https://support.huawei.com/enterprise/en/doc/index.html

[19] Huawei. 2025. Ascend 910B. https://https://e.huawei.com/cn/products/computing/ascend

[20] Huawei. 2025. Kunpeng 920. https://www.hikunpeng.com/compute/kunpeng920

[21] Huawei. 2025. OceanStor Dorado V6. https://e.huawei.com/en/products/storage/all-flash-storage/dorado-8000-18000-v6

[22] Huawei. 2025. OceanStor Dorado V6 Series 6.1.x and V700R001 SmartTier Feature Guide. https://support.huawei.com/enterprise/en/doc/EDOC1100214961/426cffd9/about-this-document

[23] Ilias Iliadis, Jens Jelitto, Yusik Kim, Slavisa Sarafijanovic, and Vinodh Venkatesan. 2017. ExaPlan: Efficient Queueing-Based Data Placement, Provisioning, and Load Balancing for Large Tiered Storage Systems. *ACM Trans. Storage* 13, 2 (2017), 17:1–17:41.

[24] Juwon Kim, Minsu Kim, Muhammad Danish Tehseen, Joontaek Oh, and Youjip Won. 2022. IPLFS: Log-Structured File System without Garbage Collection. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. Carlsbad, CA, 739–754.

[25] Juwon Kim, Seungjae Lee, Joontaek Oh, Dongkun Shin, and Youjip Won. 2025. D2FS: Device-Driven Filesystem Garbage Collection. In *23rd USENIX Conference on File and Storage Technologies, FAST 2025, Santa Clara, CA, February 25-27, 2025*. USENIX Association, 337–353.

[26] Jaehyung Kim, Hongchan Roh, and Sanghyun Park. 2018. Selective I/O Bypass and Load Balancing Method for Write-Through SSD Caching in Big Data Analytics. *IEEE Trans. Comput.* 67, 4 (2018), 589–595. https://doi.org/10.1109/TC.2017.2771491

[27] K. R. Krish, Bharti Wadhwa, and etc. 2016. On Efficient Hierarchical Storage for Big Data Processing. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. 403–408. https://doi.org/10.1109/CCGrid.2016.61

[28] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. 2020. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 514–528. https://doi.org/10.1145/3387514.3406591

[29] Mayuresh Kunjir, Brandon Fain, Kamesh Munagala, and Shivnath Babu. 2017. ROBUS: Fair Cache Allocation for Data-parallel Workloads. 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, 219–234. https://doi.org/10.1145/3035918.3064018

[30] Junghee Lee, Youngjae Kim, Galen M. Shipman, Sarp Oral, and Jongman Kim. 2013. Preemptible I/O Scheduling of Garbage Collection for Solid State Drives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 2 (2013), 247–260.

[31] Tania Panayiotou, Konstantinos Manousakis, Sotirios P Chatzis, and Georgios Ellinas. 2019. A data-driven bandwidth allocation framework with QoS considerations for EONs. *Journal of Lightwave Technology* 37, 9 (2019), 1853–1864.

[32] Jinting Ren, Xianzhang Chen, and etc. 2019. Archivist: A Machine Learning Assisted Data Placement Mechanism for Hybrid Storage Systems. In *37th IEEE International Conference on Computer Design, ICCD 2019, Abu Dhabi, United Arab Emirates, November 17-20, 2019*. IEEE, 676–679.

[33] Yujie Ren, David Domingo, Jian Zhang, and etc. 2025. PolyStore: Exploiting Combined Capabilities of Heterogeneous Storage. In *23rd USENIX Conference on File and Storage Technologies, FAST 2025, Santa Clara, CA, February 25-27, 2025*. USENIX Association, 539–555.

[34] Mohit Saxena, Michael M. Swift, and Yiying Zhang. 2012. FlashTier: a lightweight, consistent and durable storage cache. In *European Conference on Computer Systems, Proceedings of the Seventh EuroSys Conference 2012, EuroSys '12, Bern, Switzerland, April 10-13, 2012*. ACM, 267–280.

[35] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE* 104, 1 (2015), 148–175.

[36] Jiajie Shen, Yi Li, Yangfan Zhou, and Xin Wang. 2019. Understanding I/O performance of IPFS storage: a client's perspective. In *Proceedings of the International Symposium on Quality of Service* (Phoenix, Arizona) *(IWQoS '19)*. Association for Computing Machinery, New York, NY, USA, Article 17, 10 pages.

[37] Gagandeep Singh, Rakesh Nadig, Jisung Park, and etc. 2022. Sibyl: adaptive and extensible data placement in hybrid storage systems using online reinforcement learning. In *ISCA '22: The 49th Annual International Symposium on Computer Architecture, New York, New York, USA, June 18 - 22, 2022*. ACM, 320–336.

[38] Zhenhua Tan, Linbo Long, Jingcheng Shen, Renping Liu, Congming Gao, Kan Zhong, and Yi Jiang. 2024. Optimizing Garbage Collection for ZNS SSDs via In-storage Data Migration and Address Remapping. *ACM Trans. Archit. Code Optim.* 21, 4 (2024), 77:1–77:25.

[39] Leandros Tassiulas and Anthony Ephremides. 1990. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. In *29th IEEE Conference on Decision and Control*. IEEE, 2130–2132.

[40] K. Villalobos, V.J. Ramírez-Durán, B. Diez, J.M. Blanco, A. Goñi, and A. Illarramendi. 2020. A three level hierarchical architecture for an efficient storage of industry 4.0 data. *Computers in Industry* 121 (2020), 103257.

[41] Hui Wang and Peter J. Varman. 2014. Balancing fairness and efficiency in tiered storage systems with bottleneck-aware allocation. In *Proceedings of the 12th USENIX conference on File and Storage Technologies, FAST 2014, Santa Clara, CA, USA, February 17-20, 2014*. USENIX, 229–242.

[42] Junshe Wang and Pengzhou Shi. 2018. Research on Bandwidth Control Technology Based on SDN. In *2018 2nd IEEE Advanced Information Management,Communicates,Electronic and Automation Control Conference (IMCEC)*. 705–708. https://doi.org/10.1109/IMCEC.2018.8469269

[43] Weimeng Wang, Lei Liu, and Zhongmin Yan. 2023. Proactive Auto-scaling For Delay-sensitive Service Providers Over Public Clouds. In *2023 IEEE/ACM 31st*

International Symposium on Quality of Service (IWQoS). 01–09. https://doi.org/10.1109/IWQoS57198.2023.10188731

[44] Kan Wu, Zhihan Guo, Guanzhou Hu, Kaiwei Tu, Ramnatthan Alagappan, Rathijit Sen, Kwanghyun Park, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. 2021. The storage hierarchy is not a hierarchy: Optimizing caching on modern storage devices with orthus. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*. 307–323.

[45] XiaoJian Wu and A. L. Narasimha Reddy. 2010. Exploiting Concurrency to Improve Latency and throughput in a Hybrid Storage System. In *MASCOTS 2010, 18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Miami, Florida, USA, August 17-19, 2010*. IEEE Computer Society, 14–23.

[46] Wenjian Xiao, Huanqing Dong, Liuying Ma, Zhenjun Liu, and Qiang Zhang. 2016. HS-BAS: A hybrid storage system based on band awareness of Shingled Write Disk. In *34th IEEE International Conference on Computer Design, ICCD 2016, Scottsdale, AZ, USA, October 2-5, 2016*. IEEE Computer Society, 64–71.

[47] Minghao Ye, Yang Hu, Junjie Zhang, Zehua Guo, and H. Jonathan Chao. 2023. Reinforcement Learning-based Traffic Engineering for QoS Provisioning and Load Balancing. In *2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*. 1–10. https://doi.org/10.1109/IWQoS57198.2023.10188762

[48] Yuexi Yin, Zirui Zhuang, Jingyu Wang, Qi Qi, Haifeng Sun, Xiaoyuan Fu, and Jianxin Liao. 2024. Beyond Throughput-Optimal: Second-Order Smooth Backpressure Algorithm for Reducing Jitter and Delay. In *2024 IEEE/ACM 32nd International Symposium on Quality of Service (IWQoS)*. 1–10. https://doi.org/10.1109/IWQoS61813.2024.10682945

[49] Gong Zhang, Lawrence Chiu, Clem Dickey, Ling Liu, Paul Muench, and Sangeetha Seshadri. 2010. Automated lookahead data migration in SSD-enabled multi-tiered storage systems. In *IEEE 26th Symposium on Mass Storage Systems and Technologies, MSST 2012, Lake Tahoe, Nevada, USA, May 3-7, 2010*. IEEE Computer Society, 1–6.

[50] Ji Zhang, Xijun Li, Xiyao Zhou, Mingxuan Yuan, Zhuo Cheng, Keji Huang, and Yifan Li. 2022. L-QoCo: learning to optimize cache capacity overloading in storage systems. In *Proceedings of the 59th ACM/IEEE Design Automation Conference* (San Francisco, California) *(DAC '22)*. Association for Computing Machinery, New York, NY, USA, 379–384. https://doi.org/10.1145/3489517.3530466

[51] Tianru Zhang, Andreas Hellander, and Salman Toor. 2023. Efficient Hierarchical Storage Management Empowered by Reinforcement Learning. *IEEE Transactions on Knowledge and Data Engineering* 35, 6 (2023), 5780–5793. https://doi.org/10.1109/TKDE.2022.3176753