MINOS: Towards a Theoretical WAF = 1 in Log-Structured Storage

Xiaojun Guo[1], Guangjie Xing[1], Hua Wang[1*], Ke Zhou[1], Min Fu[2],
Ming Xie[2], Fenqiang Yang[2], Guangchao Yang[2], Jingyu Tang[1], and Jinhui Xu[1]
[1]*Huazhong University of Science and Technology (HUST)*
[2]*Tencent Technology (Shenzhen) Co., Ltd.*
**Corresponding Author: Hua Wang (hwang@hust.edu.cn)*

Abstract

Separating data by invalidation time is critical for mitigating garbage collection (GC) overhead in log-structured storage. In this paper, we establish a formal model and derive an OP-Minimized Oracle that optimizes separation thresholds for a theoretical Write Amplification Factor (WAF) of 1 under limited physical streams. Guided by this Oracle, we then design MINOS (MINimized Over-provisioning Separation), a practical online approximation that classifies writes into fine-grained virtual streams, tracks their invalidation behaviors, and adaptively clusters them into limited physical streams. Across 170 workloads totaling 500 TiB of writes, MINOS reduces GC writes by 15.0%–40.5% and improves throughput by up to 14.2% over state-of-the-art approaches, while incurring negligible CPU and memory overhead.

1 Introduction

Log-structured storage, a design that performs all writes in an append-only manner, is a foundational architecture in modern storage systems, from device firmwares [16, 32, 54], file systems [18, 24, 28, 47], distributed storage systems [6, 31], to key-value stores [34, 42]. This design is compatible with sequential write-friendly hardware [5, 37, 67], thereby delivering stable performance [19, 29] while simplifying consistency management [6, 8]. However, its reliance on out-of-place updates renders prior data versions obsolete, which then occupy physical space as invalid data. This necessitates a garbage collection (GC) process to reclaim space [5, 47].

The GC process operates at a coarse, segment-level granularity and must migrate any remaining valid data from a victim segment before the entire segment can be reused. This migration leads to write amplification: the system performs more writes than issued by the user. The magnitude of this effect is measured by the Write Amplification Factor (WAF), defined as: $WAF = \frac{\text{User Writes+GC Writes}}{\text{User Writes}}$. High WAF degrades critical system metrics: it shortens device lifespan by consuming program-erase cycles [36, 39, 67], throttles throughput by congesting internal bandwidth [14, 31, 53, 65], and harms Quality of Service (QoS) by introducing resource contention [9, 21, 22, 46, 55]. WAF is governed by multiple factors, including workload characteristics [17, 24, 61, 62], available system resources like over-provisioning (OP) space [5, 35, 55], and crucial algorithmic choices such as the data separation strategy [7, 10, 25, 39, 39, 41, 44, 50, 51, 53, 59, 60] and victim selection policy [2, 13, 33, 34, 47].

Data separation has emerged as the most effective algorithmic technique for controlling WAF. Modern log-structured systems provide explicit mechanisms to support this [5, 28, 67]. For instance, the F2FS file system employs multi-head logging [28] to direct hot and cold data into distinct logs. We refer to these limited, system-level separation channels as *physical streams*. The key insight behind state-of-the-art (SOTA) data separation techniques is to co-locate data by its *invalidation time* [7, 41, 51, 53]. The invalidation time of a data block is defined as the volume of subsequent logical writes that will occur before the block is updated. These SOTA approaches typically involve a two-phase process: *prediction* of invalidation times, followed by a *separation* phase that uses thresholds to group data with similar predicted invalidation times.

Oracle-based policies have been developed to explore the theoretical performance bounds of this approach by assuming perfect foresight of invalidation times [7, 27, 41, 53]. However, while these policies assume perfection in the prediction phase, their separation phase often relies on predefined heuristics or necessitates excessive hardware resources. Typical methods in the separation phase include using fixed thresholds based on segment size [53], partitioning the invalidation time distribution into equal-sized quantiles [7], or applying K-means clustering [41]. Recently, the **g**rouping by **l**ife**t**ime (GLT) [27] rigorously established the first theoretical WAF upper bound near unity, but its *separation* phase requires the number of streams to scale linearly with the over-provisioning. In practice, this can necessitate hundreds of streams, significantly exceeding the physical stream constraints of some modern storage devices, which typically support only 4–14 hardware separation channels [5, 45, 64].

We address these limitations by first establishing a for-

mal model that quantifies the relationship between OP space, workload invalidation characteristics, GC cost and separation thresholds. Based on this model, we propose the OP-Minimized Oracle policy, which solves the following question: *how can separation thresholds be optimized to achieve a near WAF* $= 1$ *for a given workload while minimizing the required OP space under a fixed number of physical streams?* Quantitatively, compared to prior heuristic Oracle-based policies [7, 41, 53], our method reduces GC writes by 10.6%–100% under identical OP constraints. Furthermore, compared to SOTA analytical policy GLT [27], our policy reduces the number of physical streams required to reach WAF ≈ 1 by up to 90%, demonstrating significantly higher efficiency under realistic hardware constraints.

However, directly implementing our OP-Minimized Oracle policy in practice faces three major challenges: the impact of *Prediction Error* inherent to any real-world predictor [7, 51]; the need for an efficient *Online Separation* mechanism that can find effective separation without global foreknowledge [64]; and the strict *Overhead* constraints for CPU and memory on the critical I/O path [41, 47].

To address these challenges, we present MINOS (MINimized Over-provisioning Separation), a lightweight and adaptive data separation technique designed to approximate our Oracle-based policy's objective as a practical online proxy. MINOS leverages a two-level stream management framework that distinguishes between numerous logical *virtual streams* [64] and the limited, system-level physical streams. Incoming data is first classified into a virtual stream based on its predicted invalidation time. Subsequently, an adaptive clustering process maps these virtual streams to physical streams, thereby determining data co-location.

This architecture allows us to systematically address the three challenges. To handle *Prediction Error*, its virtual streams serve a dual role: they enhance forecast accuracy with *correlative prediction* while simultaneously tracking the actual invalidation distribution of each virtual stream via our novel *Invalidation Distribution Vector (ID-Vector)*. This ensures that final separation decisions are based on reliable, observed behavior, not faulty initial predictions. To achieve *Online Separation*, MINOS periodically clusters virtual streams into physical streams through a process formally guided by our Oracle's objective, replacing the heuristics of prior work [7, 41, 51, 53, 64]. The entire design maintains low *Overhead* by keeping the critical write path simple, accelerating infrequent clustering with SIMD (Single Instruction, Multiple Data) instructions, and minimizing its memory footprint through *Spatial Metadata Aggregation*.

In summary, our contributions are as follows:

1. **Formal Modeling:** We establish a model quantifying the interplay between OP, workloads, GC cost, and separation thresholds for Oracle based policies. We propose OP-Minimized Oracle policy to optimize separation thresholds under limited physical streams to achieve near WAF $= 1$ with minimal OP space (§ 3).

2. **System Design:** We design MINOS, which employs a formally-guided clustering mechanism that approximates the Oracle's objective as a practical online proxy. MINOS addresses the challenges of *Prediction Error*, *Online Separation* and low *Overhead* (§ 4).

3. **Evaluation:** We evaluate MINOS across 170 cloud workloads (500 TiB writes), ensuring statistical rigor via Batch Means Analysis [15]. MINOS reduces GC writes by 15.0%–40.5% and boosts throughput by up to 14.2% over SOTA systems (§ 5).

Concurrent work. DOGI [23] is concurrent work on Oracle-guided data placement, targeting a similar problem to ours. Unlike our theory-guided formulation and optimization under limited physical streams, its design is built on exhaustive trace-driven calibrated Oracle policy. We separately evaluate MINOS against DOGI's released artifact(§ 5.5).

2 Background and Related Work

2.1 WAF in Log-Structured Storage

Log-structured storage systems operate by appending all incoming data blocks to large, fixed-size *segments*. This out-of-place-update design means that new data is always written to an *open* segment, rendering previous versions of the data invalid. Once full, a segment becomes *sealed* and immutable.

The accumulation of this invalid data consumes physical space. Systems manage this using two complementary mechanisms: a pre-allocated buffer of *OP* space to accommodate the invalid data [5], and a *GC* mechanism to reclaim space. GC is triggered when the amount of free space drops below a critical threshold [41, 53]. The GC mechanism selects a *victim* segment from the sealed segments, migrates its valid data to open segments, and erases the victim segment for reuse.

This internal data migration during GC is the source of write amplification. The magnitude of WAF is governed by several factors, including workload characteristics [17, 24, 61, 62], system parameters like segment size and OP ratio [5, 35, 53, 55], and, most critically, the policies for *data separation* [7, 10, 25, 39, 39, 41, 44, 50, 51, 53, 59, 60] and *victim selection* [2, 13, 33, 34, 47]. As system parameters are typically fixed, research has largely focused on designing more effective data separation and victim selection policies.

2.2 Data Separation Techniques

Separating data by update characteristics is a fundamental approach to reducing WAF in log-structured storage [39]. Modern systems expose explicit support for such separation, ranging from multi-head logging in F2FS [28] to hardware

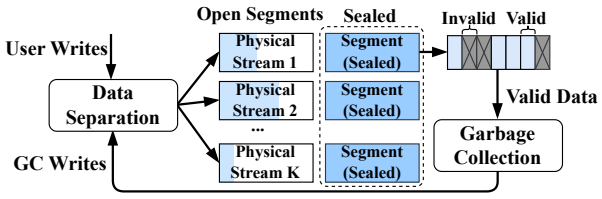


Figure 1: The workflow of a general data separation scheme.

interfaces such as ZNS SSDs [5,40] and FDP [1,45]. We refer to these system-level separation channels as *physical streams*, as shown in Figure 1.

Theoretical work has progressively improved our understanding of how data should be separated. Early analytic models focused on update frequency or access skewness [13, 35, 43, 61], while more recent studies derive Oracle-based bounds based on invalidation-time order or lifetime under the assumption that the Oracle knows exactly when each written block will be invalidated [26, 27]. These studies characterize how an ideal data placement can reduce GC cost, and recent results such as GLT [27] establish WAF bounds near 1. However, such analyses typically do not target the realistic setting of a small, fixed number of physical streams. In particular, GLT attains the bounds based on the assumption that the number of streams scales with the number of segments.

System designs have also adopted Oracle-guided insights to reduce WAF. A common pattern is to first construct Oracle-style offline policies under perfect invalidation-time knowledge, in order to study how lower-WAF placement should group data with similar invalidation behaviors, and then use the gained insights to guide practical system designs. Representative Oracle-style grouping policies include fixed segment-size thresholds in *Segment Oracle* [53], equal-sized quantiles in *Percentile Oracle* [7], K-means clustering in *K-Means Oracle* [41], and exhaustive trace-driven calibration in DOGI's NoDaP [23]. These Oracle-guided explorations have inspired practical designs that predict invalidation time and manage grouping online. However, commodity devices expose only a small number of physical streams (e.g., 4–14) [5, 45, 64], and existing Oracle-style policies still rely on heuristic or empirically calibrated grouping rules rather than a formal objective for separation-threshold optimization.

Our work addresses the problem on two fronts. First, we introduce the OP-Minimized Oracle, which formulates separation-threshold optimization explicitly under a limited number of physical streams. We then design MINOS, a practical online realization of this objective. Like vStream [64], MINOS uses a two-level architecture with virtual and physical streams, but its virtual-to-physical mapping is guided by a formal objective derived from our model.

2.3 Victim Selection Policy

Two simple victim selection policies are *First-In-First-Out (FIFO)* [13, 34] and *Greedy* [2, 47, 57]. *Cost-Benefit (CB)* policy [47, 48] is often more effective for real workloads, as it balances the short-term benefits of reclaiming space and the long-term costs. The typical CB formula is $value = \frac{gp}{1-gp} \times age$, where the segment with the highest value is selected for GC. Here, gp represents the proportion of invalid data in the segment, and age represents the logical time since the sealing of the segment. Some studies attempt to enhance the victim selection policies, proposing different formulas [11, 33].

In this paper, we employ the CB policy as the default to evaluate the performance of data separation policies as it generally outperforms FIFO and Greedy [41].

3 A Formal Model for Data Separation

This section develops a model to select separation thresholds, with the goal of minimizing the required OP space to achieve an ideal WAF of 1. We first quantify the inefficient trade-off between WAF and OP in single-stream systems (§ 3.1) and show how data separation improves it (§ 3.2). Then we present our theoretical OP-Minimized Oracle (§ 3.3), which derives the separation thresholds. Finally, we outline the real-world challenges that motivate the design of MINOS (§ 3.4).

3.1 Trade-Off between WAF and OP Space

To formalize the well-established trade-off between WAF and OP space [5, 13, 35, 53, 55], we model an idealized log-structured system. This system consists of two tiers: a write tier (Tier 1) with a finite capacity of T segments, and an infinite archival tier (Tier 2). When Tier 1 becomes full, a GC process reclaims a victim segment by migrating its valid data to Tier 2. The core of our analysis is the workload's invalidation time CDF, denoted as $F(t)$, where t represents logical time in units of segments written. We ground our analysis in a running example using an i.i.d. Zipf distribution workload [62] ($\theta = 1.01$), as shown in Figure 2.

Assuming a single-stream system, the invalidation-time CDF enables a direct calculation of WAF. When Tier 1 reaches its capacity T, the oldest segment has an age of T. In this idealized model, this segment is the expected reclamation choice for FIFO, Greedy, and CB policies, as it has the highest estimated invalidation ratio, $F(T)$. The fraction of valid data in this segment is therefore $1 - F(T)$. Reclaiming this segment requires migrating its valid data to the GC-free Tier 2, yielding a WAF of $1 + (1 - F(T))$. For instance, under this abstraction, a Tier 1 capacity of $T = 1000$ corresponds to $F(1000) \approx 0.8$, yielding a WAF of 1.2.

Furthermore, the model can quantify the total volume of invalid data across all segments in Tier 1, which corresponds to the OP space. This volume is given by the sum of the

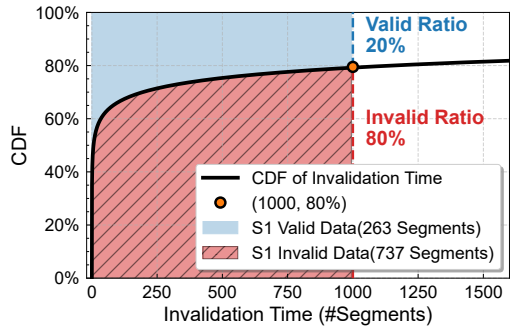


Figure 2: The invalidation time CDF $F(t)$ for a Zipf distribution $\theta = 1.01$ workload of 1TiB of data writes on 100 GiB device generated by FIO [4]. The segment size is set to 32 MiB for the analysis of this workload.

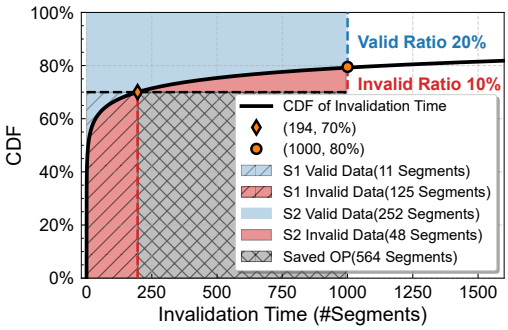


Figure 3: OP space reduction via two-stream separation.

invalid data of segments with ages from 0 to T, which can be formulated as an integral: $\int_0^T F(t)\mathrm{d}t$. As visualized in Figure 2 for $T = 1000$, this integral is the red-shaded area, amounting to 737 segments, while the blue area holds the remaining 263 segments of valid data.

This analytical result reveals an inefficient trade-off in single-stream systems: GC write $(1 - F(T))$ decreases sublinearly while the required OP space grows almost linearly $(\int_0^T F(t)\mathrm{d}t)$, indicating severe diminishing returns.

3.2 Reducing OP Space with Data Separation

Data separation can fundamentally alleviate the WAF-OP trade-off, significantly reducing the OP space requirement for a given target WAF. We illustrate this using a two-stream configuration that separates data based on a single invalidation time threshold, τ. Data with an invalidation time less than τ is directed to a hot stream (S1), while all other data is assigned to a cold stream (S2).

Let's analyze the same Zipf workload with a target $WAF = 1.2$, which required 1000 segments of capacity in the single-stream system. We set the separation threshold $\tau = 194$ segments, where $F(194) \approx 70\%$, as depicted in Figure 3.

- **Hot Stream (S1):** This stream receives 70% of writes. If segments are reclaimed at an age of 1000, S1 would require $70\% \times 1000 = 700$ segments of space. Because all data in S1 is guaranteed to be invalid within 194 segment-writes, we only need to buffer these segments for 194 time units. This strategy requires $70\% \times 194 \approx 136$ segments and achieves a WAF of 1 for this stream.
- **Cold Stream (S2):** This stream receives the remaining 30% of writes. For this analytical example, we fix its reclamation age at 1000, matching the single-stream baseline. S2 therefore requires $30\% \times 1000 = 300$ segments, and the system-wide 20% valid-data migration overhead is attributed entirely to this stream.

Algorithm 1: Finding Thresholds for OP-Minimized Oracle Policy

Function `CalOP`(τ, F)
  `// Calculates total OP for WAF=1`
  $OP_{\text{total}} \leftarrow 0$;
  for $i = 1 \to K$ **do**
    $OP_{\text{stream}} \leftarrow \int_{\tau_{i-1}}^{\tau_i} (F(t) - F(\tau_{i-1}))\mathrm{d}t$;
    $OP_{\text{total}} \leftarrow OP_{\text{total}} + OP_{\text{stream}}$;
  return OP_{total};

Input: Invalidation time CDF $F(t)$, stream count K.
Output: A vector of $K-1$ thresholds $(\tau_1, \ldots, \tau_{K-1})$.

$\tau = (\tau_0, \tau_1, \ldots, \tau_{K-1}, \tau_K)$ such that $F(\tau_i) = i/K$;
$OP_{\text{min}} \leftarrow$ `CalOP`(τ, F);
converged $\leftarrow$ false;
while ***not*** *converged* **do**
  `// Optimize` τ `with coordinate descent`
  converged $\leftarrow$ true;
  for $i = 1 \to K-1$ **do**
    $\tau_i' \leftarrow \arg\min_{t \in (\tau_{i-1}, \tau_{i+1})}$ `CalOP`$(\tau|_{\tau_i = t}, F)$;
    $OP_{\text{new}} \leftarrow$ `CalOP`$(\tau|_{\tau_i = \tau_i'}, F)$;
    if $OP_{new} < OP_{\text{min}}$ **then**
      $\tau_i \leftarrow \tau_i'$ `// Update the threshold`
      $OP_{\text{min}} \leftarrow OP_{\text{new}}$;
      converged $\leftarrow$ false;
return $(\tau_1, \ldots, \tau_{K-1})$;

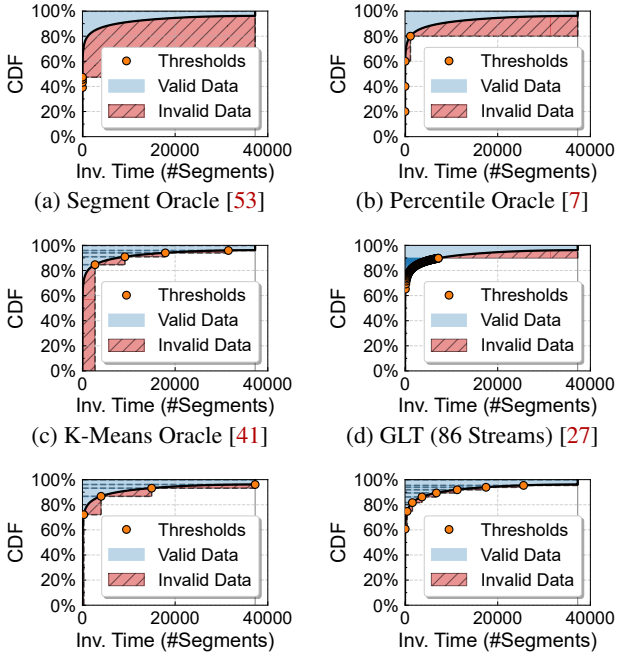
Consequently, the total space required by this two-stream system is reduced to $136 + 300 = 436$ segments, while achieving the same target WAF of 1.2 as the 1000-segment single-stream baseline. As visualized in Figure 3, this principle works by enabling zero-cost reclamation in the hot stream, thereby eliminating the OP space (gray-shaded area) required by the single-stream system and saving 564 segments.

3.3 The OP-Minimized Oracle policy

The preceding analysis demonstrates that the invalidation-time CDF model provides a geometric framework to quantify

Table 1: Comparison of different policies across various workloads and OP ratios. Except for Desnoyers's Model, which is computed analytically, all reported WAFs are obtained from trace-driven simulations. Our approach achieves optimal WAF (= **1**) comparable to GLT [27] but requires significantly fewer streams (e.g., fixed 10 vs. up to 161 for GLT).

Policy \ Workload (OP)	Uniform				Zipf-1.01				Backend-Tencent				Backend-Ali			
	15%	20%	25%	30%	15%	20%	25%	30%	15%	20%	25%	30%	15%	20%	25%	30%
Desnoyers's [13]	3.419	2.636	2.161	1.848	1.249	1.166	1.114	1.080	1.237	1.176	1.136	1.105	1.484	1.328	1.230	1.169
Desnoyers's Model [13]	3.491	2.681	2.110	1.870	1.902	1.592	1.389	1.276	1.487	1.315	1.216	1.153	1.997	1.651	1.450	1.319
Segment Oracle [53]	3.460	2.654	2.170	1.853	1.357	1.284	1.232	1.193	1.215	1.180	1.154	1.135	1.457	1.356	1.284	1.228
Percentile Oracle [7]	1.620	1.362	1.203	1.103	1.225	1.164	1.121	1.089	1.169	1.130	1.102	1.078	1.259	1.184	1.133	1.097
K-Means Oracle [41]	1.584	1.311	1.152	1.050	1.072	1.051	1.036	1.024	1.041	1.033	1.026	1.021	1.091	1.060	1.037	1.020
Ours (6 Streams)	1.522	1.273	1.125	1.034	1.030	1.009	1.001	**1**	1.009	1.001	1.000	**1**	1.066	1.024	1.006	1.000
GLT [27]	1.246	1.001	**1**	**1**	1.033	1.003	**1**	**1**	1.042	1.011	1.003	**1**	1.058	1.022	1.001	**1**
– *req. Streams*	(49)	(68)	(90)	(116)	(86)	(121)	(161)	(206)	(62)	(87)	(115)	(147)	(49)	(68)	(91)	(115)
Ours (10 Streams)	1.147	1.011	**1**	**1**	1.000	**1**	**1**	**1**	1.000	**1**	**1**	**1**	1.003	1.000	**1**	**1**



(a) Segment Oracle [53] (b) Percentile Oracle [7]

(c) K-Means Oracle [41] (d) GLT (86 Streams) [27]

(e) OP-Min. Oracle (6 Streams) (f) OP-Min. Oracle (10 Streams)

Figure 4: Separation thresholds for different Oracle-based policies under Zipf 1.01 workload. The red shaded area represents the required OP space to achieve $WAF = 1$.

the trade-off between GC overhead, OP space, and separation thresholds. Building on this, we propose the *OP-Minimized Oracle*, a formal policy that optimizes separation thresholds under the constraint of limited physical streams. Unlike prior Oracle-based policies [7, 27, 41, 53], our policy explicitly optimizes the required OP space needed to achieve a theoretical $WAF = 1$ for a fixed, small number of streams K.

As detailed in Algorithm 1, the policy establishes an objective function, `CalOP`, which models the required OP space as the total integrated volume of buffered invalid data across all streams based on the workload's CDF. The policy then employs *coordinate descent* (Lines 9–17) to iteratively refine the $K-1$ separation thresholds for K streams. This optimization is significant because the area calculated by `CalOP` quantifies the OP necessary to ensure that all invalid data is buffered until segments become completely invalid, thereby guaranteeing zero-cost garbage collection.

To validate our approach, we conduct a comparative simulation using synthetic and real-world cloud workloads [30, 66] under OP constraints, and measure the total WAF. We compare our policy against three distinct pillars: the frequency-based model by Desnoyers [13], heuristic-based limited stream policies [7, 41, 53] and the analytically derived, unbounded-stream GLT policy [27]. For a fair assessment of practical utility, following [53], we configure our policy and other heuristic-based Oracles (e.g., Segment, K-Means, Percentile) with a standard six-stream constraint (5 for user and 1 for GC), while also allowing our policy to be tested in a ten-stream configuration for comparison with GLT. The separation thresholds are also visualized in Figure 4.

The results in Table 1 show the advantages of our approach in three key perspectives. ① While Desnoyers's model [13] effectively predicts WAF for uniform workloads, policies based on invalidation time achieve significantly lower WAFs, confirming that invalidation time is a more fundamental metric for optimizing data placement than update frequency. ② Within the standard six-stream constraint, our policy consistently outperforms heuristic-based Oracles, reducing GC writes ($WAF - 1$) by 10.6%–100%. For instance, on the Zipf workload with 20% OP, our approach reduces GC writes to 0.009, significantly outperforming the next-best K-Means Oracle (0.051). ③ While GLT [27] can achieve $WAF = 1$, it demands an impractical number of streams for commercial hardware, such as 161 open streams for the Zipf workload. In contrast, our Oracle achieves this near $WAF = 1$ with only 10 streams, demonstrating that analytically optimizing thresh-

olds for a limited stream budget can attain SOTA performance with greater practical feasibility.

Optimization Scope. Algorithm 1 uses coordinate descent and returns a coordinate-wise converged solution, without guaranteeing global optimality for arbitrary invalidation-time CDFs. It minimizes the OP required for $WAF = 1$, rather than WAF under a fixed OP; therefore, when $OP < OP_{\min}$, its thresholds are not guaranteed to be WAF-optimal.

3.4 The Gap Between Theory and Practice

However, translating the OP-Minimized Oracle policy into a deployable system presents three primary challenges:

Prediction Error. The assumption of perfect foresight as Oracle is unrealistic. While prior work has focused on improving prediction accuracy [41, 51], we argue that the more fundamental challenge is *not merely to reduce prediction errors, but to build a system that is resilient to their consequences*.

Online Separation. Our Oracle-based policy requires *a priori* knowledge of the entire, static invalidation time distribution to find optimized thresholds, which is untenable for online systems. Prior approaches have resorted to suboptimal compromises: static, offline profiling that cannot adapt to workload changes [7]; simplistic heuristics that fail on complex distributions [51, 53]; or model-driven searching that are brittle to workload shifts [41]. The challenge, therefore, is to make effective separation decisions online and adaptively, without global foreknowledge or exhaustive search.

Overhead. Data separation executes on the critical I/O path, so it cannot become a performance bottleneck. This constraint demands a lightweight design with minimal CPU and memory footprints, precluding heavyweight solutions that require host offloading or hardware accelerators (e.g., GPUs) [7, 51].

4 MINOS Design

To address these challenges, we introduce MINOS, a practical technique designed to bridge the gap between our theoretical model and real-world constraints. Specifically, MINOS adopts the optimization objective of the OP-Minimized Oracle as an efficient proxy for online WAF reduction. It realizes the feasibility via transforming the complex continuous threshold search into a tractable discrete problem via a two-level stream architecture. As illustrated in Figure 5, the workflow comprises four key stages:

❶ ***Virtual stream prediction*** (§ 4.1). MINOS classifies all writes (from both user and GC) into fine-grained virtual streams [64], each representing an invalidation-time interval.

❷ ***Actual invalidation time tracking*** (§ 4.2). To mitigate ***Prediction Error***, MINOS tracks the actual invalidation time distribution for each virtual stream using our novel ID-Vector.

❸ ***Adaptive stream clustering*** (§ 4.3). MINOS periodically clusters these discrete virtual streams into physical streams based on the actual invalidation time distributions captured in their ID-Vectors. This clustering process is formally guided by the objective of our OP-Minimized Oracle, enabling effective ***Online Separation***.

❹ ***Physical stream placement.*** Based on the current clustering decisions, data blocks are written to the open segment corresponding to their assigned physical stream.

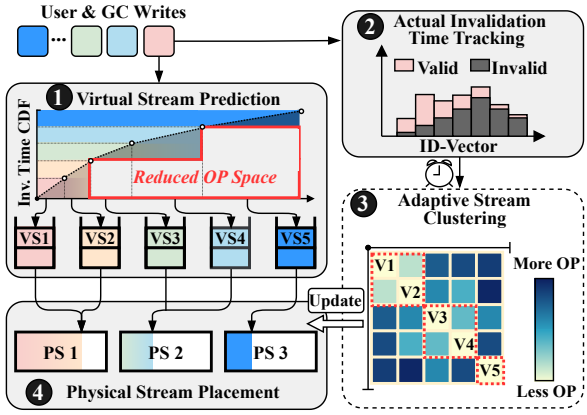


Figure 5: Overview of MINOS.

The MINOS design is engineered for low ***Overhead*** (§ 4.4). The per-write critical path is kept simple (❶→❷→❹), and the adaptive clustering (❸) is also highly optimized. We minimize memory footprint with *Spatial Metadata Aggregation*, accelerate clustering with *SIMD instructions*, and tune parameters to achieve high performance and low runtime cost.

4.1 Virtual Stream Prediction

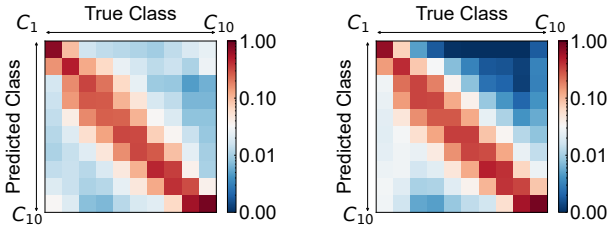
MINOS partitions the observed invalidation-time CDF into equal-probability intervals, with each interval represented by a virtual stream. Assigning a write to a virtual stream therefore classifies its next invalidation time into one of these intervals. This equal quantile-based partitioning balances the traffic across virtual streams, thus creating a stable input for the subsequent clustering stage. We maintain the interval boundaries online using the P^2 algorithm [20], which has $O(1)$ time and space complexity.

Algorithm 2 uses the elapsed time since a block's last user write as a lightweight prediction signal [41, 51, 53]. New user writes are assigned to a dedicated stream ($N_{user} - 1$) because their invalidation behavior is initially unknown (Line 4). For data updates ($[0, N_{user} - 2]$), *correlative prediction* moves the current virtual-stream ID by at most one step toward the target indicated by the last invalidation time (Lines 6–9). As shown in Figure 6, among writes assigned to C_1, the fraction whose actual invalidation times exceed the C_1 interval decreases from 22.48% to 11.15%, reducing cold-as-hot misclassification. GC writes use a separate estimator and dedicated virtual streams ($[N_{user}, N_{user} + N_{gc} - 1]$), isolating them from user writes (Lines 13–16).

Algorithm 2: Virtual Stream Prediction

Input: Metadata B, current time t_{curr}, I/O type $isUser$
Output: Assigned virtual stream ID id_{vs}

$\Delta t \leftarrow t_{curr} - B.lastUserWriteTime$;
if $isUser$ **then**
  if $B.lastUserWriteTime = INITIAL_VALUE$ **then**
    $id_{vs} \leftarrow N_{user} - 1$;
    `// New writes use the last user stream`
  else
    `Update(`$userCDF, \Delta t$`)`;
    $id_{target} \leftarrow$`Query(`$userCDF, \Delta t$`)`;
    $id_{vs} \leftarrow B.lastUserVsId$;
    $id_{vs} \leftarrow id_{vs} -$`sign(`$id_{vs} - id_{target}$`)`;
    `// Gradually adjust ID by one step`
  `// Update block metadata`
  $B.lastUserWriteTime \leftarrow t_{curr}$;
  $B.lastUserVsId \leftarrow id_{vs}$;
else
  `// Classify GC writes using` Δt
  `Update(`$gcCDF, \Delta t$`)`;
  $id_{vs} \leftarrow N_{user} +$`Query(`$gcCDF, \Delta t$`)`;
  $B.lastGCWriteTime \leftarrow t_{curr}$;
  $B.lastGCVsId \leftarrow id_{vs}$;
return id_{vs};



(a) w/o Correlative Prediction (b) w/ Correlative Prediction

Figure 6: Confusion matrices for prediction accuracy on the Tencent cloud workload, where invalidation times are uniformly discretized into 10 classes (C_1 to C_{10}).

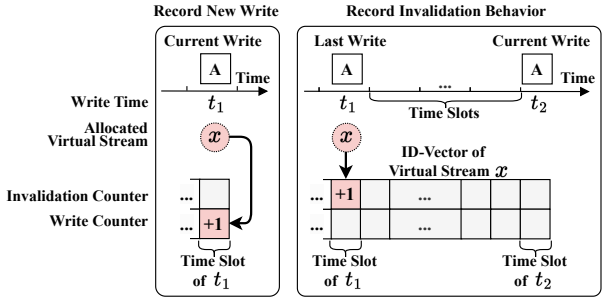


Figure 7: ID-Vector records the *write counter* and *invalidation counter* of virtual streams.

4.2 Actual Invalidation Time Tracking

Predictive outputs are inevitably accompanied by errors for learning-augmented online algorithms [27]. We argue that the critical challenge is not to endlessly pursue prediction accuracy with more powerful and costly models [7, 51], but rather to design a system that makes robust decisions in the presence of these errors. To this end, MINOS introduces the ID-Vector to record the actual invalidation distribution for each virtual stream, providing ground truth feedback that is essential for our clustering process.

MINOS divides logical time into time slots, each spanning the write volume of a single segment. For each time slot, every ID-Vector maintains two counters: a *Write Counter* recording the amount of data written within that slot, and an *Invalidation Counter* tracking the amount of data invalidated. Figure 7 illustrates this process: when a block A is written to virtual stream x at time slot t_1, the Write Counter $W_x^{t_1}$ is incremented. When that block is later invalidated, MINOS increments the Invalidation Counter $I_x^{t_1}$ for the block's *original* write slot t_1. To manage memory overhead, these counters are maintained in a circular array of a fixed length L.

This ID-Vector design offers two advantages over the prior Update Interval Distribution [41]. First, it provides a direct measurement of invalid data on a per-time-slot basis. By tracking writes that are issued together, it enables a precise calculation of the OP space required to buffer them. Second, because all ID-Vectors share a common time basis, their counters can be *arithmetically combined* via element-wise summation to model the invalidation distribution for any proposed virtual stream cluster. These two properties are essential for enabling effective, OP-minimizing clustering decisions.

4.3 Adaptive Stream Clustering

To bridge the gap between our theoretical objective (minimizing global OP) and online execution, MINOS employs adaptive stream clustering. The objective is to map N virtual streams into K physical streams such that the total "Marginal OP Cost" incurred by merging dissimilar data is minimized.

The Distance Metric. The core of our clustering logic is a "mixing test", as detailed in Figure 8. Consider two virtual streams, x and y, with their respective written volumes (W_x^t, W_y^t) and invalid data volumes (I_x^t, I_y^t) tracked by the ID-Vector. If these streams were merged into a single physical stream, their combined invalidation ratio would be $R_{x+y}^t = \frac{I_x^t + I_y^t}{W_x^t + W_y^t}$. If we theoretically mix x and y and then sample a volume equivalent to W_x^t from this mixture, the expected amount of invalid data in that sample, denoted as $I_{x|x+y}^t$, is:

$$I_{x|x+y}^t = \left(\frac{I_x^t + I_y^t}{W_x^t + W_y^t} \right) \cdot W_x^t. \quad (1)$$

This equation quantifies how the presence of stream y "pol-

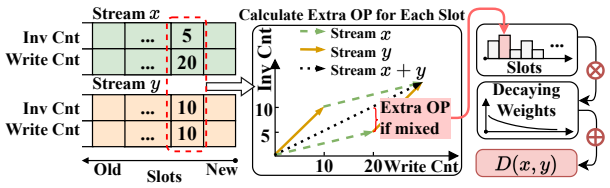


Figure 8: MINOS quantifies the distance between virtual streams as the additional OP required.

Algorithm 3: Virtual Stream Clustering

Input: Set of N virtual streams' ID-Vector V, Number of physical streams K
Output: Mapping of virtual streams to physical streams M

1 Initialize each virtual stream as a cluster $C_1, \cdots, C_N$;
2 **while** *Number of clusters* $> K$ **do**
3 Find the two **adjacent clusters** C_i and C_j with the minimum distance $D(i,j)$;
4 $V[i] = V[i] + V[j]$;
5 Merge C_j into C_i;
6 $id_{phy} = 1$;
7 **for** *cluster C in remaining clusters* **do**
8 **for** *virtual stream id i in C* **do**
9 $M[i] = id_{phy}$;
10 id_{phy} += 1;
11 **return** M;

lutes" the reclamation efficiency of stream x. The Marginal OP Cost ($D_t(x,y)$) at time slot t is then defined as the total deviation from their isolated states:

$$D_t(x,y) = |I^t_{new} - I^t_x| = \frac{|W^t_x I^t_y - W^t_y I^t_x|}{W^t_x + W^t_y}. \quad (2)$$

The total distance $D(x,y)$ is a weighted sum of these per-slot distances over the ID-Vector's history length L. We employ exponentially decaying weights to prioritize recent workload behavior, making the MINOS more responsive to changes. The final distance metric is:

$$D(x,y) = \sum_{t=1}^{L} D_t(x,y) \cdot \alpha^{t-1}, \quad (3)$$

where α is a decay factor, which is set to $1 - \frac{1}{L}$.

Efficient Hierarchical Clustering. We employ a modified hierarchical clustering algorithm as detailed in Algorithm 3. The process begins with each of the $N = N_{user} + N_{gc}$ virtual streams as its own cluster and iteratively merges the pair with the minimum distance until the target number of physical streams, K, is reached. To make this process highly efficient, we introduce two key optimizations.

First, the algorithm exclusively merges *adjacent* clusters (Line 3). This heuristic leverages the fact that our virtual streams are already ordered by predicted invalidation time. After each merge, only two new distances need to be re-calculated. This limits the total number of distance calculations to $O(N)$. Second, each distance calculation operates on ID-Vectors of length L, and when clusters are merged, their ID-Vectors are combined via element-wise summation (Line 4), an operation that takes $O(L)$ time. Combining these two optimizations, the overall time complexity of the clustering process is $O(N \cdot L)$.

4.4 Overhead and Parameterization

We detail how MINOS reduces its memory and computational overhead and discuss the selection of its key parameters.

Memory Overhead Mitigation. MINOS records four metadata fields per block: the last user-write and GC-write timestamps (32 bits each, measured in segment-write slots), and their corresponding virtual-stream IDs (4 bits each). These fields require 9 bytes per 4 KiB block, or 2.25× the size of a standard 4-byte mapping pointer [67].

To address this, we introduce *Spatial Metadata Aggregation*. This technique leverages the spatial locality of physically contiguous data by storing a single shared metadata entry for each group of eight adjacent blocks, which was also explored in prior designs [41, 64]. This optimization reduces the average metadata overhead to 9 bits per block, an 87.5% reduction from the naive design with negligible impact on performance.

SIMD-Enhanced Clustering. To ensure our highly efficient clustering algorithm introduces no I/O path latency spikes, we further optimize its core operations using *SIMD* instructions. The underlying ID-Vector arithmetic is fundamentally vector-based and thus highly amenable to data parallelism. Leveraging SIMD is not a reliance on specialized hardware but an efficient use of standard, on-die capabilities found in modern server CPUs (e.g., AVX2) and enterprise SSD controllers (e.g., ARM NEON [3]). Our intra-core optimization is therefore preferable to heavyweight offloading approaches [51] or designs requiring a background core [41], ensuring it does not impact QoS.

Parameterization. With these optimizations in place, we configure MINOS's four key parameters: the number of user virtual streams (N_{user}), GC virtual streams (N_{gc}), the ID-Vector history length (L), and the reclustering interval ($W_{recluster}$). Based on a comprehensive parameter sweep (§ 5.2), we chose a default configuration of $N_{user} = 12$, $N_{gc} = 4$, an L equal to the total number of segments in the system (typically thousands [1, 5]), and a $W_{recluster}$ of 8 segments. As our evaluation will show, this configuration strikes a robust balance between WAF reduction and low overhead.

5 Evaluation

This section comprehensively evaluates MINOS by answering three key questions: (1) How effectively does it reduce

Table 2: Evaluated workloads.

Workload	Time	#Volume	Working set size (GiB)	Write traffic (TiB)	Average write size (KiB)
Vol-Tencent [66]	2018	62	15425	148	71.4
Vol-Ali [30]	2020	105	18459	357	52.6
BE-Tencent [66]	2018	-	1022	27.7	34.5
BE-Ali [30]	2020	-	803	16.6	17.1
BE-Cloud	2024	-	1071	10.5	66.9
Zipf-H [4]	Zipf($\theta = 1.01$)		340	4	64
Zipf-C [4]	Zipf($\theta = 0.80$)		340	4	64

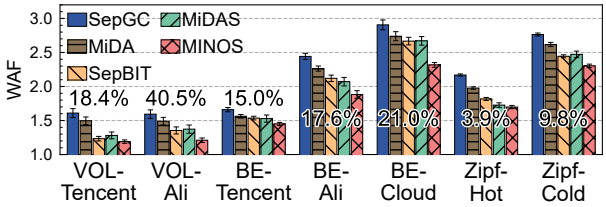


Figure 9: WAF under default configuration.

WAF compared to SOTA policies? (§ 5.2) (2) What are the resulting end-to-end performance gains? (§ 5.3) (3) What is its overhead? (§ 5.4)

5.1 Experimental Setup

Platforms. We evaluate MINOS's performance on two platforms: a trace-driven simulator and a real-world, log-structured storage system. Our simulator, based on the framework from MiDAS [41], evaluates the WAF of different data separation policies across a wide range of workloads and configurations. To ensure statistical validity, we discard the first 30% of the trace to reach a steady state and employ Batch Means Analysis [15] using 30 batches to calculate the 95% confidence intervals. For system-level evaluation, we implement MINOS and all baselines in CSAL [67], a modern log-structured block storage system built on SPDK [63]. CSAL's high throughput enables rigorous stress-testing and accurate measurement of each policy's overhead.

Hardware and System Configuration. System experiments are conducted on a server with an Intel Xeon Platinum 8255C CPU, 128 GB of DRAM, and a 7.68 TiB Samsung PM1733 NVMe SSD. We adjust the physical SSD space available to CSAL to match the Working Set Size (WSS) of each workload. The default configuration uses a 512 MiB segment size and a 15% OP space, consistent with prior work [53].

Baselines. We compare MINOS against a spectrum of policies, ranging from low-overhead heuristics to the SOTA. Legacy approaches [10, 25, 39, 49, 50, 59, 60] are excluded as SepBIT [53] and MiDAS [41] has already proven to outperform them.

- **SepGC** [52]: A minimal-overhead policy that isolates user writes from GC writes.
- **MiDA** [44]: An extension of SepGC that further separates GC writes based on data migration counts.
- **SepBIT** [53]: An advanced policy that uses a fixed six-stream model (1 hot user, 1 cold user, 4 GC) to isolate user and GC writes with different invalidation times.
- **MiDAS** [41]: The SOTA policy. It employs a Markov model to derive a static configuration optimized for stationary workloads. We evaluate it to test the resilience of this model-first approach against the dynamic, real-world traces central to our study.

To ensure a fair comparison, all baseline policies are configured with a maximum of six physical streams and the CB victim selection policy. MiDAS is the sole exception to both rules; to faithfully replicate its design, it is allowed to autotune its stream count (up to 20) and employs its specified hybrid victim selection policy (FIFO for non-final groups, CB for the final group). Consistent with the MiDAS methodology, our primary metric is the reduction in GC writes ($WAF - 1$). While this work focuses on a low-overhead design, our separation framework is orthogonal to and can complement DNN-based predictors [7, 51].

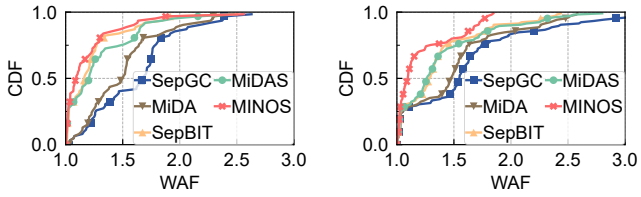
Workloads. Our evaluation employs a diverse set of workloads detailed in Table 2, with a focus on challenging, real-world traces. We also include synthetic workloads to facilitate a comprehensive and fair comparison against prior art that performs well under such idealized conditions.

- **Real-World Traces:** We select large-scale, open-source cloud volume traces from Tencent Cloud [66] and Alibaba Cloud [30], focusing on volumes with large working sets ($\geq$50 GiB) and high write churn (traffic $\geq 2\times$ WSS) (VOL-Tencent, VOL-Ali). We also use aggregated backend traces (BE-Tencent, BE-Ali, BE-Cloud). These are crucial as they faithfully represent the complex, interleaved I/O patterns from multiple tenants found in production cloud storage [56].
- **Synthetic Traces:** We use FIO [4] to generate two Zipf workloads: one hot (Zipf-H, $\theta = 1.01$) and one cool (Zipf-C, $\theta = 0.8$).

We also evaluate MINOS in DOGI's released artifact (§ 5.5), since it cannot be fully integrated into our framework.

5.2 WAF Reduction and Policy Analysis

Overall Performance. Figure 9 presents the average WAF of MINOS and the baselines across seven workloads. MINOS consistently outperforms the baselines, reducing GC writes by 15.0%–40.5% on real-world workloads and by 3.9%–9.8% on the Zipf workloads. However, under the same segment size and OP configuration, the six-stream OP-Minimized Oracle



(a) Tencent Volume Workload (b) Ali Volume Workload

Figure 10: WAF CDF under different volume workloads.

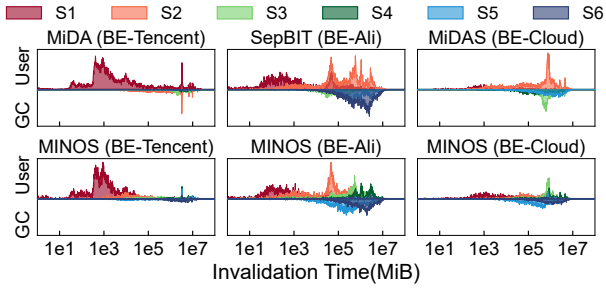


Figure 11: Invalidation time distributions across various streams under the backend workloads.

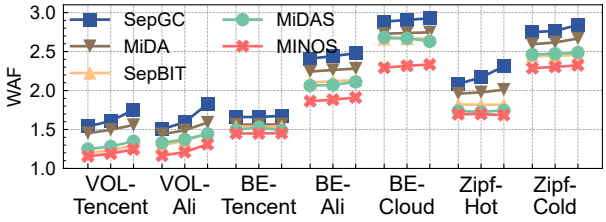


Figure 12: WAF sensitivity to segment size. Within each workload group, points from left to right represent sizes of 256, 512, and 1024 MiB.

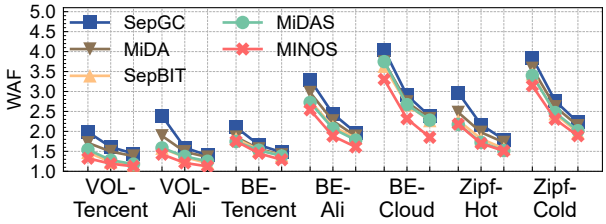


Figure 13: WAF sensitivity to the OP ratio. Within each workload group, points from left to right represent OP ratios of 10%, 15%, and 20%.

achieves WAFs of 1.009 and 1.066 on BE-Tencent and BE-Ali, respectively (Table 1), revealing a substantial gap between the practical online policy and the offline Oracle.

Performance of Volume Workloads. To provide a more granular view, Figure 10 shows the CDF of WAF across all individual volumes for the two volume workloads. For the Tencent workload, MINOS reduces the median GC writes by 43.2% compared to the next-best policy, SepBIT. This advantage is more pronounced on the challenging Alibaba workload. While data separation provides little benefit for approximately 25% of these volumes, MINOS dramatically improves performance for the remainder, reducing the median GC writes by 64.6% relative to SepBIT.

Analysis of Data Separation Efficiency. MINOS's superior performance stems from its more effective data separation, as illustrated in Figure 11. On BE-Ali, SepBIT's reliance on only the last invalidation time leads to significant misclassification in its hot stream S1 around 10^5 MiB. On BE-Cloud, MiDAS mixes a wide range of invalidation times in its cold stream S2, with a notable peak around 10^6 MiB that is quickly migrated to the GC stream S3. In contrast, MINOS uses its correlative prediction to purify its streams and its adaptive clustering to isolate specific invalidation times. As a result, MINOS mitigates the inefficient peaks of GC writes seen in the baselines, leading to a substantial reduction in WAF.

Sensitivity to System Parameters. We now evaluate the robustness of MINOS by analyzing its sensitivity to three critical system parameters: segment size, OP ratio, and the number of physical streams.

Impact of Segment Size. We vary the segment size from 256 MiB to 1 GiB, as shown in Figure 12. A smaller segment size reduces WAF by lowering GC granularity. MINOS's relative advantage grows as segment size decreases, with its average reduction in GC writes on real-world traces increasing from 18.4% at 1 GiB to 23.7% at 256 MiB.

Impact of OP Ratio. MINOS's performance advantage grows with the OP ratio. As shown in Figure 13, raising the OP from 10% to 20% increases its average reduction in GC writes on real-world traces from 16.0% to 27.9%.

Impact of Number of Physical Streams. MINOS effectively leverages additional physical streams, demonstrating the scalability of its adaptive clustering. As shown in Figure 14, increasing the stream count from six to ten (the maximum streams MiDAS required for the Zipf-C workload) achieves a finer-grained data separation, further reducing GC writes by 6.7% to 13.2% on large-scale backend and synthetic workloads compared to the default six-stream configuration.

Parameter Sensitivity and Robustness Analysis. Figure 15 illustrates the performance of MINOS across various configurations and prediction noises, with results normalized to the SOTA GC write reduction.

Parameter Sensitivity. We evaluate MINOS's sensitivity to stream count (N_{user} and N_{gc}), history length (L), and reclustering interval ($W_{recluster}$). We set $N_{user} = 12$ and $N_{gc} = 4$ as further increases yield diminishing returns. L is set to the total system segment count, as longer histories reduce adaptability. MINOS remains stable across various reclustering intervals, and we set the default $W_{recluster}$ to 8 segment-writes.

Component Contribution. Replacing Adjacent Clustering

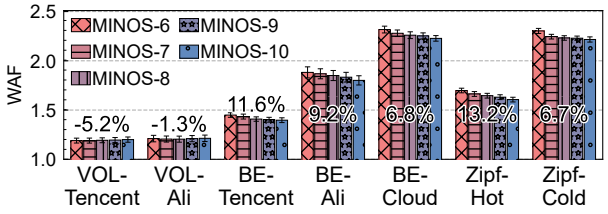


Figure 14: Impact of the number of streams in the system.

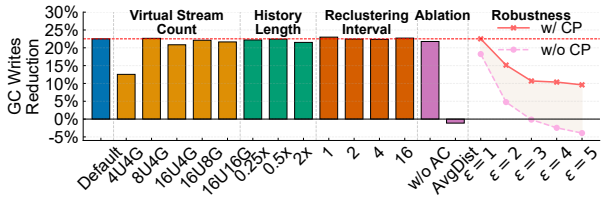


Figure 15: Sensitivity analysis of MINOS's key algorithmic parameters on GC writes Reduction.

(w/o AC) with a full search yields no discernible gain, justifying our localized approach. Conversely, substituting our guided clustering with a simple average invalidation time metric (AvgDist, similar to vStream [64]) causes significant degradation, confirming that formal guided data separation is the core driver of efficiency.

Robustness to Prediction Noise. To assess Correlative Prediction (CP), we introduce a multiplicative noise factor ε, scaling predicted invalidation times by a random variable within $[1/\varepsilon, \varepsilon]$. As shown by the solid line, MINOS remains resilient, maintaining nearly 10% reduction even at $\varepsilon = 5$. In contrast, the baseline without CP (dashed line) is highly vulnerable, with performance falling below zero when $\varepsilon \geq 3$. This underscores the critical safety margin provided by our CP mechanism.

Adaptability to Dynamic Workloads. To evaluate MINOS's adaptability to non-stationary workloads, we designed a 10 TiB dynamic workload that alternates between Zipf-H and Zipf-C distributions every 1 TiB of writes. As shown in Figure 16, MINOS consistently maintains the lowest and most stable WAF. This stability stems from MINOS's dynamic clustering mechanism, which continuously monitors the invalidation behaviors of each virtual stream.

Specifically, MINOS evaluates the distance between streams based on Equation 3. When a clustering operation is triggered, MINOS calculates the distances between adjacent streams and merges the closest pair to minimize clustering-induced overhead. For example, as visualized in Figure 17, at 7.5 TiB (Zipf-C), the system faces high GC pressure; in response, MINOS merges the statistically similar GC streams 14 and 15. In contrast, at 8.5 TiB, under the Zipf-H workload, GC pressure lessens, and the first two GC virtual streams become statistically similar. MINOS again adapts by merging

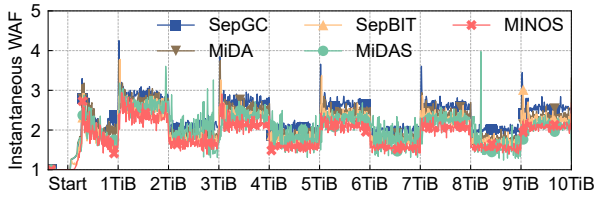
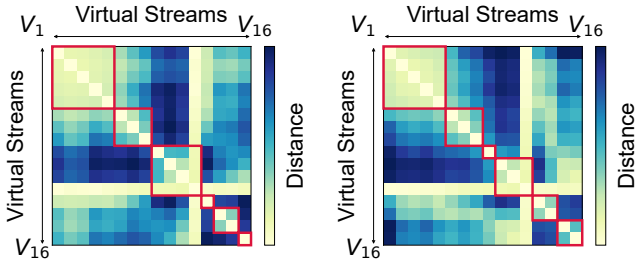


Figure 16: Instantaneous WAF under a dynamic workload that alternates between Zipf-H and Zipf-C every 1 TiB.



(a) Clustering at 7.5TiB(Zipf-C) (b) Clustering at 8.5TiB(Zipf-H)

Figure 17: Clustering results of MINOS.

them and reallocating the freed physical stream to user writes.

In contrast, MiDAS experiences WAF fluctuations due to its reliance on a model-derived configuration. When the workload shifts to Zipf-H, its previously optimal 9-stream configuration (established during the Zipf-C phase) requires time to adjust. Because the newly formed hot-data groups have not yet reached their required size, MiDAS temporarily selects colder groups for GC. This reclaims low-utility segments, leading to transient WAF increases until the system accumulates enough data to trigger a reconfiguration.

5.3 System Performance

We now evaluate the end-to-end performance of MINOS in our CSAL implementation to demonstrate how its WAF reduction translates into system-level benefits. All experiments run in a single-threaded polling mode with a dedicated, fully-saturated CPU core to handle all I/O and data separation logic. All workloads are replayed using FIO with a queue depth of 128 to saturate the storage device.

Throughput and WAF. Figure 18 presents the average throughput and WAF for each policy. As expected, the WAF in the real system is generally higher than in simulation because GC is triggered reactively by space watermarks. Nonetheless, MINOS consistently achieves a lower WAF than all baselines. Compared to the next-best policy, MINOS reduces GC writes by an average of 13.5%, with a peak reduction of 19.5% on the BE-Cloud workload.

This reduction in background I/O directly translates to higher throughput. By minimizing GC interference, MINOS

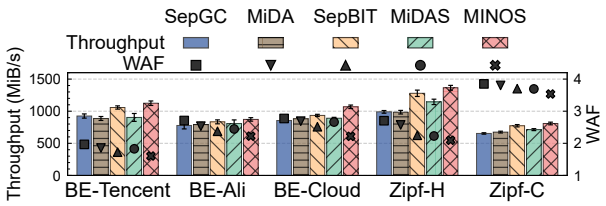
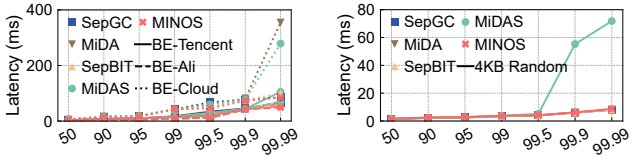


Figure 18: WAF and throughput for CSAL [67] under different workloads and data separation policies.



(a) Backend workloads. (b) 4KB random write.

Figure 19: Latency distribution for different workloads.

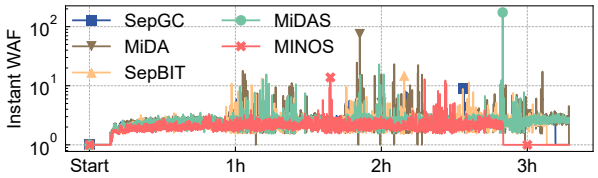


Figure 20: Instantaneous WAF of different data separation policies over time under the BE-Cloud workload.

increases average throughput by 7.7% over the next-best policy, with a maximum improvement of 14.2% on the BE-Cloud workload. These results confirm that our design's efficiency gains are not merely theoretical but deliver significant real-world performance improvements.

Latency and QoS. Figure 19a presents the write latency for the backend workloads. MINOS maintains excellent latency and QoS, with a profile nearly identical to SepBIT, the best-performing lightweight baseline. Across all backend workloads, its latency at all percentiles is consistently within a tight margin ($< 103\%$) of SepBIT's. This is a crucial result, demonstrating that our method improves overall system efficiency without compromising QoS on the critical I/O path. In contrast, MiDAS and MiDA suffer from significantly higher tail latency. This is a direct consequence of the periodic WAF spikes (visualized in Figure 20) caused by their static policies failing to adapt, a behavior analyzed in §5.2.

5.4 Overhead Analysis

Computational Overhead. We analyze computational overhead from two angles: the per-I/O overhead on the critical path and the latency impact of the periodic clustering task.

To measure the impact of our per-I/O logic, we use a 4 KiB random write workload where throughput directly reflects software path efficiency. As shown in Table 3, MINOS is exceptionally lightweight, achieving 81.7K IOPS, a throughput comparable to SepBIT and SepGC. In contrast, MiDAS's throughput drops by 6.2% to 76.6K IOPS. As corroborated by Figure 19b, its design also causes severe tail latency spikes.

For the periodic clustering task, our SIMD optimization proves highly effective, as shown in Table 4. The results demonstrate a speedup of approximately 10×, with latency reaching just over 100μs even for a large system with 8192 segments. This scale is highly relevant, as the fundamental reclaim unit in modern storage systems is now multi-gigabyte in size (e.g., 1 GiB ZNS zones, 6 GiB FDP Reclaim Units, and 67.4 GiB super blocks) [1, 5, 67]. Consequently, efficiently managing over 8000 such segments demonstrates MINOS's scalability to systems with tens of terabytes of capacity. At a cost comparable to a single SSD I/O operation, this infrequent task has a negligible impact on overall performance.

Memory Overhead. Table 5 shows the trade-off of Spatial Metadata Aggregation. WAF is averaged over the real-world workloads in our main benchmark suite, and AggN means that N contiguous blocks share one metadata entry. Our default Agg8 uses 288 MiB with a negligible WAF increase (less than 0.004). MiDAS uses 74 MiB by sampling 1% of blocks and storing a 2-bit hot filter per block. Under a near iso-memory setting, Agg32 still achieves substantially lower WAF, demonstrating the efficiency of our theory-guided design.

Table 3: IOPS under a 4KB random write workload.

Policy	SepGC	MiDA	MiDAS	SepBIT	**MINOS**
IOPS (K)	81.6	83.9	76.6	82.8	**81.7**

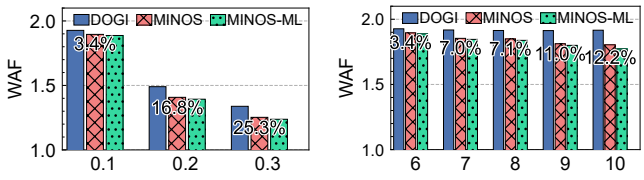
Table 4: Computational overhead of clustering.

#Segment	512	1024	2048	4096	8192
w/o SIMD (μs)	75.0	145.8	306.9	590.5	1167.9
w/ SIMD (μs)	6.3	12.1	23.1	49.4	117.9

Table 5: Memory overhead and average WAF comparison.

Policy	**Memory(MiB/TiB)**	**Average WAF**
MINOS (NoAgg)	2304	1.606
MINOS (Agg8, Default)	288	1.610
MINOS (Agg32)	72	1.621
SepGC	∼0	2.042
MiDA	∼0	1.910
SepBIT	590*	1.782
MiDAS	74	1.785

*SepBIT's overhead is based on the realistic snapshot-case analysis from its original paper [53].



(a) OP ratio. (b) Physical streams.

Figure 21: Comparison with DOGI on its open-source artifact.

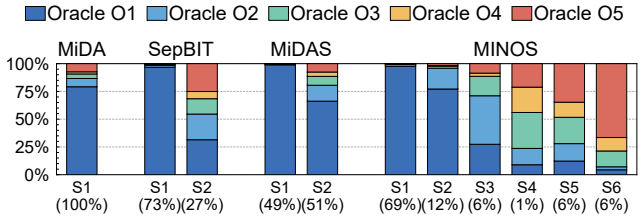


Figure 22: Oracle composition of user-write placement on BE-Tencent. Only user writes are shown. Colors denote Oracle assignments, and percentages below the bars denote write shares.

5.5 Evaluation on DOGI's Artifact

DOGI [23] releases only its implementation and a Zipf-based workload, without the full evaluation code used in its paper. Therefore, we report this experiment separately from our main benchmark suite. We implement MINOS in DOGI's released framework and further integrate an MLP predictor into MINOS, denoted as MINOS-ML. Unless otherwise stated, we use DOGI's default artifact settings (segment size 64 MB, OP=10%, and $K = 6$ physical streams).

As shown in Figure 21, MINOS consistently achieves lower WAF than DOGI on the released Zipf workload. Its advantage grows with higher OP, reaching up to 25.3% GC-write reduction at OP=30%, and also increases with more physical streams, reaching up to 12.2% at $K = 10$. MINOS-ML further improves over MINOS, showing that better prediction is complementary to MINOS.

6 Discussion

6.1 Approximating the Offline Oracle

Although MINOS consistently outperforms existing online policies, its WAF remains higher than that of the OP-Minimized Oracle under the same configuration (Section 5.2). To characterize this gap, Figure 22 compares how the user writes placed in each physical stream would be grouped by the Oracle on BE-Tencent. Compared with MiDA, SepBIT, and MiDAS, MINOS's separation of user writes more closely matches the Oracle, while the remaining cross-Oracle mixing indicates that it does not fully reproduce the offline separation.

6.2 Connection to Classical Locality Theory

Classical locality studies characterize a workload's WSS and cache behavior using reuse time (the number of intervening accesses between two consecutive references to the same data item) and stack distance [12, 38]. The Higher Order Theory of Locality (HOTL) unifies these metrics into a hierarchy [58]. In its continuous form, given a reuse-time CDF $F_r(t)$, the time-window miss ratio $m(t)$ and mean WSS $w(T)$ satisfy

$$\begin{aligned} m(t) &= 1 - F_r(t), \\ w(T) &= \int_0^T m(t)\,\mathrm{d}t. \end{aligned} \tag{4}$$

Section 3.1 similarly uses the invalidation-time CDF $F(t)$ to characterize the required OP space and GC cost in a log-structured storage system. Invalidation time, measured as the volume of intervening logical writes before the next update, is the write-side counterpart of reuse time. Thus, $1 - F(T)$ determines the fraction of valid data that must be migrated during GC, while $\int_0^T F(t)\,\mathrm{d}t$ determines the invalid-data volume occupying OP space.

We believe this correspondence can help bring more locality-analysis tools into the study of data separation in log-structured storage.

7 Conclusion

Efficient data separation is critical to log-structured storage. We introduce the OP-Minimized Oracle, which formulates threshold selection under limited physical streams with the objective of minimizing the OP required for a theoretical WAF of 1. Guided by this objective, MINOS provides a lightweight and adaptive online technique. Evaluations show that MINOS reduces GC writes by 15.0%–40.5% and improves throughput by up to 14.2% over SOTA approaches, demonstrating the practical value of formally guided data separation.

Acknowledgments

This work was supported by the National Key Research and Development Program of China under Grant No. 2023YFB4502701, and the National Natural Science Foundation of China under Grant Nos. 62232007 and 62172180. We are grateful to our shepherd, Sungjin Lee, and the anonymous reviewers for their comments and suggestions.

References

[1] Michael Allison, Arun George, Javier Gonzalez, Dan Helmick, Vikash Kumar, Roshan R Nair, and Vivek Shah. Towards efficient flash caches with emerging nvme flexible data placement ssds. In *Proceedings of the Twenti-*

eth European Conference on Computer Systems, pages 1142–1160, 2025.

[2] Ernst Althaus, Petra Berenbrink, André Brinkmann, and Rebecca Steiner. On the optimality of the greedy garbage collection strategy for ssds. In *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, pages 78–88. IEEE, 2022.

[3] Arm Limited. Powerful real-time processor for smart storage applications. https://www.arm.com/products/silicon-ip-cpu/cortex-r/cortex-r82, 2025. Accessed: September 2, 2025.

[4] Jens Axboe. Fio: Flexible i/o tester. https://github.com/axboe/fio, 2025. Accessed: January 10, 2025.

[5] Matias Bjørling, Abutalib Aghayev, Hans Holmberg, Aravind Ramesh, Damien Le Moal, Gregory R Ganger, and George Amvrosiadis. ZNS: Avoiding the block interface tax for flash-based SSDs. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 689–703, 2021.

[6] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, et al. Windows azure storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 143–157, 2011.

[7] Chandranil Chakraborttii and Heiner Litz. Reducing write amplification in flash by death-time prediction of logical block addresses. In *Proceedings of the 14th ACM International Conference on Systems and Storage*, pages 1–12, 2021.

[8] Jianjun Chen, Yonghua Ding, Ye Liu, Fangshi Li, Li Zhang, Mingyi Zhang, Kui Wei, Lixun Cao, Dan Zou, Yang Liu, et al. Bytehtap: bytedance's htap system with high data freshness and strong data consistency. *Proceedings of the VLDB Endowment*, 15(12):3411–3424, 2022.

[9] Zhe Chen and Yuelong Zhao. Da-gc: A dynamic adjustment garbage collection method considering wear-leveling for ssd. In *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, pages 475–480, 2020.

[10] Mei-Ling Chiang, Paul CH Lee, and Ruei-Chuan Chang. Using data clustering to improve cleaning performance for flash memory. *Software: Practice and Experience*, 29(3):267–290, 1999.

[11] Niv Dayan, Luc Bouganim, and Philippe Bonnet. Modelling and managing ssd write-amplification. *arXiv preprint arXiv:1504.00229*, 2015.

[12] Peter J. Denning. The working set model for program behavior. *Commun. ACM*, 11(5):323–333, May 1968.

[13] Peter Desnoyers. Analytic models of ssd write performance. *ACM Trans. Storage*, 10(2), March 2014.

[14] Assaf Eisenman, Asaf Cidon, Evgenya Pergament, Or Haimovich, Ryan Stutsman, Mohammad Alizadeh, and Sachin Katti. Flashield: a hybrid key-value cache that controls flash write amplification. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 65–78, 2019.

[15] Peter W Glynn and Donald L Iglehart. Simulation output analysis using standardized time series. *Mathematics of Operations Research*, 15(1):1–16, 1990.

[16] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. Dftl: a flash translation layer employing demand-based selective caching of page-level address mappings. *Acm Sigplan Notices*, 44(3):229–240, 2009.

[17] Dan Helmick. Host workloads achieving waf==1 in an fdp ssd. https://www.sniadeveloper.org/events/agenda/session/595, 2023. Accessed: December 17, 2024.

[18] Qingda Hu, Jinglei Ren, Anirudh Badam, Jiwu Shu, and Thomas Moscibroda. Log-Structured Non-Volatile main memory. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 703–717, 2017.

[19] Joo-Young Hwang, Seokhwan Kim, Daejun Park, Yong-Gil Song, Junyoung Han, Seunghyun Choi, Sangyeun Cho, and Youjip Won. ZMS: Zone abstraction for mobile flash storage. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, pages 173–189, 2024.

[20] Raj Jain and Imrich Chlamtac. The p2 algorithm for dynamic calculation of quantiles and histograms without storing observations. *Communications of the ACM*, 28(10):1076–1085, 1985.

[21] Wonkyung Kang and Sungjoo Yoo. Dynamic management of key states for reinforcement learning-assisted garbage collection to reduce long tail latency in ssd. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[22] Wonkyung Kang and Sungjoo Yoo. q-value prediction for reinforcement learning assisted garbage collection to reduce long tail latency in ssd. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2240–2253, 2019.

[23] Jeeyun Kim, Seonggyun Oh, Jungwoo Kim, Jisung Park, Jaeho Kim, Sungjin Lee, and Sam H. Noh. Dogi: Data placement with oracle-guided insights for log-structured

systems. In *24th USENIX Conference on File and Storage Technologies (FAST 23)*, 2026.

[24] Juwon Kim, Minsu Kim, Muhammad Danish Tehseen, Joontaek Oh, and YouJip Won. IPLFS:Log-Structured file system without garbage collection. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 739–754, 2022.

[25] Kevin Kremer and André Brinkmann. Fadac: A self-adapting data classifier for flash memory. In *Proceedings of the 12th ACM International Conference on Systems and Storage*, pages 167–178, 2019.

[26] Tomer Lange, Joseph Naor, and Gala Yadgar. Ssd wear leveling with optimal guarantees. In *2024 Symposium on Simplicity in Algorithms (SOSA)*, pages 306–320. SIAM, 2024.

[27] Tomer Lange, Joseph Naor, and Gala Yadgar. Optimal ssd management with predictions. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 9(2):1–28, 2025.

[28] Changman Lee, Dongho Sim, Jooyoung Hwang, and Sangyeun Cho. F2FS: A new file system for flash storage. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pages 273–286, 2015.

[29] Jongsung Lee, Donguk Kim, and Jae W Lee. Waltz: Leveraging zone append to tighten the tail latency of lsm tree on zns ssd. *Proceedings of the VLDB Endowment*, 16(11):2884–2896, 2023.

[30] Jinhong Li, Qiuping Wang, Patrick PC Lee, and Chao Shi. An in-depth analysis of cloud block storage workloads in large-scale production. In *2020 IEEE International Symposium on Workload Characterization (IISWC)*, pages 37–47. IEEE, 2020.

[31] Qiang Li, Qiao Xiang, Yuxin Wang, Haohao Song, Ridi Wen, Wenhui Yao, Yuanyuan Dong, Shuqi Zhao, Shuo Huang, Zhaosheng Zhu, et al. More than capacity: Performance-oriented evolution of pangu in alibaba. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*, pages 331–346, 2023.

[32] Seung-Ho Lim, Hyun Jin Choi, and Kyu Ho Park. Journal remap-based ftl for journaling file system with flash memory. In *High Performance Computing and Communications: Third International Conference, HPCC 2007, Houston, USA, September 26-28, 2007. Proceedings 3*, pages 192–203. Springer, 2007.

[33] David Lomet and Chen Luo. Efficiently reclaiming space in a log structured store. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 792–803. IEEE, 2021.

[34] Lanyue Lu, Thanumalayan Sankaranarayana Pillai, Hariharan Gopalakrishnan, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. Wisckey: Separating keys from values in ssd-conscious storage. *ACM Transactions On Storage (TOS)*, 13(1):1–28, 2017.

[35] Xiang Luojie and Brian M Kurkoski. An improved analytic expression for write amplification in nand flash. In *2012 International Conference on Computing, Networking and Communications (ICNC)*, pages 497–501. IEEE, 2012.

[36] Stathis Maneas, Kaveh Mahdaviani, Tim Emami, and Bianca Schroeder. Operational characteristics of SSDs in enterprise storage systems: A Large-Scale field study. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*, pages 165–180, 2022.

[37] Adam Manzanares, Noah Watkins, Cyril Guyot, Damien LeMoal, Carlos Maltzahn, and Zvonimr Bandic. ZEA, a data management approach for SMR. In *8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16)*, 2016.

[38] Richard L. Mattson, Jan Gecsei, Donald R. Slutz, and Irving L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems journal*, 9(2):78–117, 1970.

[39] Changwoo Min, Kangnyeon Kim, Hyunjin Cho, Sang-Won Lee, and Young Ik Eom. Sfs: random write considered harmful in solid state drives. In *FAST*, volume 12, pages 1–16, 2012.

[40] Jaehong Min, Chenxingyu Zhao, Ming Liu, and Arvind Krishnamurthy. eZNS: An elastic zoned namespace for commodity ZNS SSDs. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pages 461–477, 2023.

[41] Seonggyun Oh, Jeeyun Kim, Soyoung Han, Jaeho Kim, Sungjin Lee, and Sam H Noh. MIDAS: Minimizing write amplification in Log-Structured systems through adaptive group number and size configuration. In *22nd USENIX Conference on File and Storage Technologies (FAST 24)*, pages 259–275, 2024.

[42] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. The log-structured merge-tree (LSM-tree). *Acta Informatica*, 33:351–385, 1996.

[43] Changhyun Park, Seongjin Lee, Youjip Won, and Soohan Ahn. Practical implication of analytical models for ssd write amplification. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, pages 257–262, 2017.

[44] Hyunseung Park, Eunjae Lee, Jaeho Kim, and Sam H Noh. Lightweight data lifetime classification using migration counts to improve performance and lifetime of flash-based ssds. In *Proceedings of the 12th ACM SIGOPS Asia-Pacific Workshop on Systems*, pages 25–33, 2021.

[45] Joonyeop Park and Hyeonsang Eom. Fdpvirt: Investigating the behavior of fdp ssds. In *Proceedings of the 25th International Middleware Conference: Demos, Posters and Doctoral Symposium*, pages 25–26, 2024.

[46] Yi Qin, Dan Feng, Jingning Liu, Wei Tong, and Zhiming Zhu. Dt-gc: adaptive garbage collection with dynamic thresholds for ssds. In *2014 International Conference on Cloud Computing and Big Data*, pages 182–188. IEEE, 2014.

[47] Mendel Rosenblum and John K Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems (TOCS)*, 10(1):26–52, 1992.

[48] Stephen M Rumble, Ankita Kejriwal, and John Ousterhout. Log-structured memory for DRAM-based storage. In *12th USENIX Conference on File and Storage Technologies (FAST 14)*, pages 1–16, 2014.

[49] Mansour Shafaei, Peter Desnoyers, and Jim Fitzpatrick. Write amplification reduction in Flash-BasedSSDs through Extent-Based temperature identification. In *8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16)*, 2016.

[50] Radu Stoica and Anastasia Ailamaki. Improving flash write performance by using update frequency. *Proceedings of the VLDB Endowment*, 6(9):733–744, 2013.

[51] Penghao Sun, Litong You, Shengan Zheng, Wanru Zhang, Ruoyan Ma, Jie Yang, Guanzhong Wang, Feng Zhu, Shu Li, and Linpeng Huang. Learning-based data separation for write amplification reduction in solid state drives. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.

[52] Benny Van Houdt. On the necessity of hot and cold data identification to reduce the write amplification in flash-based ssds. *Performance Evaluation*, 82:1–14, 2014.

[53] Qiuping Wang, Jinhong Li, Patrick PC Lee, Tao Ouyang, Chao Shi, and Lilong Huang. Separating data via block invalidation time inference for write amplification reduction in Log-Structured storage. In *20th USENIX Conference on File and Storage Technologies (FAST 22)*, pages 429–444, 2022.

[54] Shengzhe Wang, Zihang Lin, Suzhen Wu, Hong Jiang, Jie Zhang, and Bo Mao. Learnedftl: A learning-based page-level ftl for reducing double reads in flash-based ssds. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 616–629. IEEE, 2024.

[55] Yuhong Wen, You Zhou, Fei Wu, Shu Li, Zhenghong Wang, and Changsheng Xie. Wa-opshare: Workload-adaptive over-provisioning space allocation for multi-tenant ssds. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(11):4527–4538, 2022.

[56] Haonan Wu, Erci Xu, Ligang Wang, Yuandong Hong, Changsheng Niu, Bo Shi, Lingjun Zhu, Jinnian He, Dong Wu, Weidong Zhang, et al. Hey hey, my my, skewness is here to stay: Challenges and opportunities in cloud block store traffic. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 736–752, 2025.

[57] Michael Wu and Willy Zwaenepoel. envy: a non-volatile, main memory storage system. *ACM SIGOPS Operating Systems Review*, 28(5):86–97, 1994.

[58] Xiaoya Xiang, Chen Ding, Hao Luo, and Bin Bao. Hotl: a higher order theory of locality. In *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems*, pages 343–356, 2013.

[59] Jing Yang, Shuyi Pei, and Qing Yang. Warcip: Write amplification reduction by clustering i/o pages. In *Proceedings of the 12th ACM International Conference on Systems and Storage*, pages 155–166, 2019.

[60] Jingpei Yang, Rajinikanth Pandurangan, Changho Choi, and Vijay Balakrishnan. Autostream: Automatic stream management for multi-streamed ssds. In *Proceedings of the 10th ACM International Systems and Storage Conference*, pages 1–11, 2017.

[61] Yue Yang and Jianwen Zhu. Write amplification with write skew. In *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 406–411. IEEE, 2016.

[62] Yue Yang and Jianwen Zhu. Write skew and zipf distribution: Evidence and implications. *ACM transactions on Storage (TOS)*, 12(4):1–19, 2016.

[63] Ziye Yang, James R Harris, Benjamin Walker, Daniel Verkamp, Changpeng Liu, Cunyin Chang, Gang Cao, Jonathan Stern, Vishal Verma, and Luse E Paul. Spdk: A development kit to build high performance storage

applications. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 154–161. IEEE, 2017.

[64] Hwanjin Yong, Kisik Jeong, Joonwon Lee, and Jin-Soo Kim. vStream: Virtual stream management for multi-streamed SSDs. In *10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*, 2018.

[65] Weidong Zhang, Erci Xu, Qiuping Wang, Xiaolu Zhang, Yuesheng Gu, Zhenwei Lu, Tao Ouyang, Guanqun Dai, Wenwen Peng, Zhe Xu, et al. What's the story in EBS glory: Evolutions and lessons in building cloud block store. In *22nd USENIX Conference on File and Storage Technologies (FAST 24)*, pages 277–291, 2024.

[66] Yu Zhang, Ping Huang, Ke Zhou, Hua Wang, Jianying Hu, Yongguang Ji, and Bin Cheng. OSCA: An Online-Model based cache allocation scheme in cloud block storage systems. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 785–798, 2020.

[67] Yanbo Zhou, Erci Xu, Li Zhang, Kapil Karkra, Mariusz Barczak, Wayne Gao, Wojciech Malikowski, Mateusz Kozlowski, Łukasz Łasek, Ruiming Lu, et al. Csal: the next-gen local disks for the cloud. In *Proceedings of the Nineteenth European Conference on Computer Systems*, pages 608–623, 2024.