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

Abstract

In log-structured storage systems, separating data with different invalidation times is a cornerstone strategy for mitigating garbage collection (GC) overhead and reducing the Write Amplification Factor (WAF). State-of-the-art separation policies are guided by theoretical "Oracles" that assume perfect future knowledge. However, we identify a fundamental flaw in this approach: these Oracles themselves rely on simple, ad-hoc heuristics to define the crucial separation thresholds. This leads to suboptimal data placement and a flawed understanding of the true potential of data separation.

We address this limitation with two key contributions. First, we establish a formal model to determine the optimal separation thresholds, a problem previously unaddressed. Our model's objective is to minimize the Over-Provisioning (OP) space required to achieve a theoretical WAF of 1. Our resulting OP-Minimized Oracle demonstrates a significant theoretical reduction in GC writes, ranging from 39.6% to 100% compared to prior Oracles. Second, we introduce MINOS (MINimized Over-provisioning Separation) that closely approximates this new, theoretical target. MINOS employs a two-level architecture that first predicts data into fine-grained virtual streams and then adaptively clusters them into a limited number of physical streams, guided by our formal model. Extensive evaluations show that MINOS reduces GC writes by 19.2%-37.4% on real-world workloads and improves system throughput by up to 14.2% over state-of-the-art approaches, all with negligible 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 [14, 30, 50], file systems [17, 22, 26, 44], distributed systems [6, 29], to key-value stores [32, 39]. This design is compatible with sequential write-friendly hardware [5, 35, 62], thereby delivering stable performance [18, 27] 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, 44].

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: $WAF = \frac{\text{User Writes}+\text{GC Writes}}{\text{User Writes}}$. High WAF degrades critical system metrics: it shortens device lifespan by consuming program-erase cycles [34, 36, 62], throttles throughput by congesting internal bandwidth [13, 29, 49, 60], and harms Quality of Service (QoS) by introducing resource contention [9, 20, 21, 43, 51]. WAF is governed by multiple factors, including workload characteristics [16, 22, 56, 57], available system resources like over-provisioning (OP) space [5, 33, 51], and crucial algorithmic choices such as the data separation strategy [7, 10, 23, 36, 36, 38, 41, 46, 47, 49, 54, 55] and victim selection policy [2, 12, 31, 32, 44].

Data separation has emerged as the most effective algorithmic technique for controlling WAF. Modern log-structured systems provide explicit mechanisms to support this [5, 26, 62]. For instance, the F2FS file system employs multi-head logging [26] 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, 38, 47, 49]. 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.

To explore the theoretical limits of this approach, several studies have introduced an *Oracle* policy that can flawlessly predict the invalidation time for all incoming data [7, 38, 49]. However, these Oracle studies contain a fundamental, overlooked flaw: while they assume perfection in the prediction phase, they resort to ad-hoc heuristics for the equally critical separation phase. This includes using fixed thresholds based on segment size [49], partitioning the Cumulative Distribution Function (CDF) of invalidation times into equal-sized quantiles [7], or applying K-Means to cluster the invalidation times [38]. These ad-hoc heuristics lack a formal optimization objective, resulting in suboptimal data separation and more importantly, failing to reveal the true, achievable performance limits of data separation.

We tackle this challenge by first introducing a formal objective for setting an Oracle's separation thresholds: *how can separation thresholds be chosen to achieve a theoretical WAF of 1 for a certain workload with the minimum OP space requirement?* We then answer this question with the OP-Minimized Oracle, a method that combines our formal model of required OP space with an offline search algorithm to optimize these thresholds. Quantitatively, compared to prior SOTA Oracles [7, 38, 49], our Oracle reduces theoretical GC writes by 39.6%-100% under the same OP constraints and requires up to 20% less OP space for the same WAF target.

However, directly implementing our Oracle in practice faces three major challenges: the impact of *Prediction Error* inherent to any real-world predictor; the need for an efficient *Online Separation* mechanism that can find near-optimal separation without foreknowledge or costly search; and the strict *Overhead* constraints for CPU and memory.

To address these challenges, we present MINOS (MINimized Over-provisioning Separation), a lightweight and adaptive data separation technique to approximate our Oracle's objective online. MINOS leverages a two-level stream management framework that distinguishes between numerous logical *virtual streams* [59] 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 which data to co-locate on the storage system.

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 ad-hoc heuristics of prior work [7, 38, 47, 49, 59]. This adaptive clustering enables finding near-optimal data separation without resorting to an exhaustive search. 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. We are the first to introduce a formal objective for setting separation thresholds in log-structured storage. We propose a new theoretical Oracle based on this objective and analyze the challenges of applying it in practice (§ 3).
2. We design and implement MINOS, a practical and lightweight data separation technique. It uses a novel, formally-guided clustering mechanism to approximate our Oracle's objective online, effectively addressing the challenges of *Prediction Error*, *Online Separation* and low *Overhead* (§ 4).
3. We conduct extensive trace-driven and real-system evaluations across 170 real-world workloads, totaling over 500TiB of data writes from diverse cloud applications. Across various configurations, MINOS reduces GC writes by 19.2%-37.4% over SOTA systems, boosting system throughput by up to 14.2% (§ 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 [38, 49]. 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 [16, 22, 56, 57], system parameters like segment size and OP ratio [5, 33, 49, 51], and, most critically, the policies for *data separation* [7, 10, 23, 36, 36, 38, 41, 46, 47, 49, 54, 55] and *victim selection* [2, 12, 31, 32, 44]. 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 based on its update characteristics is a fundamental technique for reducing WAF in log-structured storage [36]. Modern systems now provide explicit mechanisms for this separation, from multi-head logging in F2FS [26] to hardware-level interfaces like Zoned Namespace (ZNS)

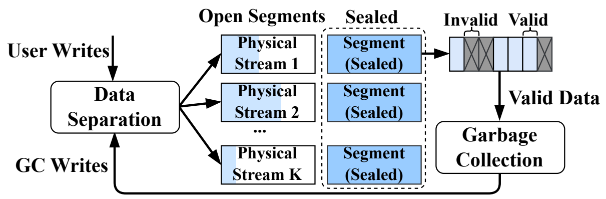


Figure 1: The workflow of a general data separation scheme.

SSDs [5, 37] and the Flexible Data Placement (FDP) standard [1,42]. We refer to these system-level separation channels as *physical streams*, as shown in Figure 1.

While early separation policies relied on indirect heuristics like update frequency [10, 23, 36, 41, 46, 48, 54, 55], SOTA methods have converged on separating data by its *invalidation time* as a more direct and effective approach [7, 38, 47, 49]. This modern approach divides the problem into two distinct phases. First, a *prediction phase* forecasts the invalidation time for incoming data, using methods ranging from simple heuristics [38, 49] to complex learning-based models [7, 47]. Second, a *separation phase* maps the predicted times to the available physical streams using a set of thresholds, such as the GC interval in SpeBIT [49] and MiDAS [38].

The design of these invalidation-time-based policies has been guided by theoretical *Oracles* that assume perfect prediction. However, a critical flaw undermines these models: while assuming perfection in prediction, their separation phase still relies on ad-hoc heuristics that lack a formal optimization objective. This reliance on heuristic separation is pervasive; while theoretical WAF models have evolved from using frequency-based proxies [12, 33, 40, 56] to deriving closed-form solutions based on true invalidation order [24] or time [25], their approach to setting separation thresholds has remained agnostic to real-world data distributions. This conceptual gap is exemplified by common heuristics such as a fixed, segment-size-based threshold as in *Segment Oracle* [49], partitioning invalidation times into equal-sized quantiles as in *Percentile Oracle* [7], or applying K-Means clustering to the invalidation times as in *K-Means Oracle* [38].

In contrast to these heuristic-based approaches, our work addresses these limitations on two fronts. First, we replace these ad-hoc Oracle models with the OP-Minimized Oracle, a new theoretical framework with a formal optimization objective for determining separation thresholds. Second, we design MINOS, a practical technique that operationalizes the insights from our theory. While MINOS utilizes a two-level stream architecture similar to vStream [59], it critically replaces the heuristic-based stream management of prior work with a principled, theory-guided clustering process.

The pursuit of formal optimization also underpins the current SOTA policy, MiDAS [38]. However, its model assumes a long-term, near-static workload. Our evaluation shows that while this model-first approach is effective on synthetic workloads, it is ill-suited for dynamic workloads, leading to performance degradation when the workload deviates from the model's assumptions.

2.3 Victim Selection Policy

Two simple victim selection policies are *First-In-First-Out (FIFO)* [12,32] and *Greedy* [2,44,53]. FIFO selects the oldest sealed segment, while Greedy selects the segment with the most invalid data. Both can be suboptimal: FIFO may select segments with cold valid data, whereas Greedy may prematurely select segments whose remaining valid data is about to be invalidated.

Cost-Benefit (CB) policy [44, 45] 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, 31].

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 [38].

3 A Formal Model for Data Separation

This section develops a formal model to determine optimal 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 reveals the optimal 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, 12, 33, 49, 51], 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 [57] ($\theta = 1.01$), 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 optimal choice for reclamation under 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 one

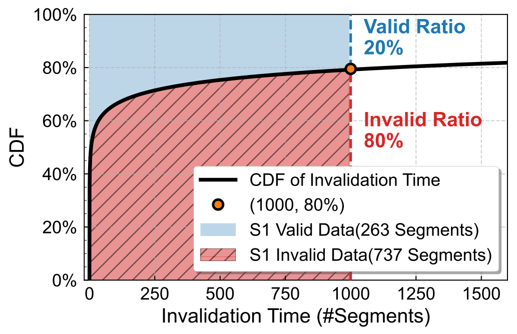


Figure 2: The invalidation time CDF $F(t)$ for a Zipf distribution $\theta = 1.01$ workload of 1TiB of data writes on 100 GiB device generated by FIO [4]. The segment size is set to 32 MiB for the analysis of this workload.

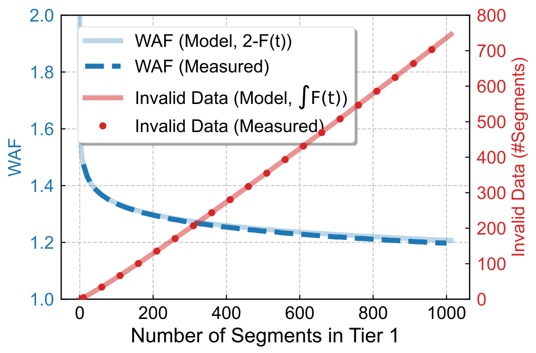


Figure 3: Validation of predicted and measured WAF and invalid data under various tier 1 capacities.

segment to make space for one incoming segment requires rewriting its valid data, incurring an instantaneous WAF of $1+(1-F(T))$. For a steady-state workload, this value represents the long-term asymptotic WAF. For instance, in Figure 2, a Tier 1 capacity of $T = 1000$ corresponds to$F(1000) \approx 0.8$, yielding an asymptotic WAF of 1.2 for the system.

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 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.

Figure 3 validates our model, showing a near-perfect alignment between its WAF and OP space predictions and experimental results across a range of Tier 1 capacities (T). This result reveals an inefficient trade-off in single-stream systems: WAF decreases sub-linearly while the required OP space grows, indicating severe diminishing returns.

3.2 Reducing OP Space with Data Separation

This subsection demonstrates how 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).

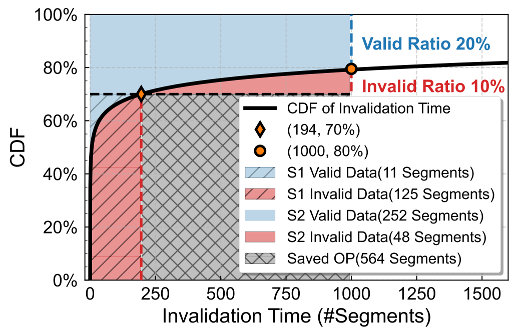


Figure 4: OP space reduction via two-stream separation.

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 4.

- **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. To maintain the overall system's target WAF of 1.2, we still reclaim S2 segments at an age of 1000, which requires a capacity of $30\% \times 1000 = 300$ segments. At this reclamation age, the system-wide 20% valid data migration overhead is now attributed solely to S2, thus maintaining the target GC overhead.

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 4, 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.

To validate our analytical model, we performed experiments with various separation thresholds (τ). In each experiment, the streams were configured to maintain an overall WAF of 1.2 (0% for S1, 20% for S2). Figure 5 shows that our model's predictions align almost perfectly with experimental results for the required OP space. The key insight is that the choice of τ has a dramatic impact on the total required OP space, even when the overall target WAF remains constant.

This analysis reveals a fundamental, two-part principle for achieving a theoretical $WAF = 1$ with minimal OP space.

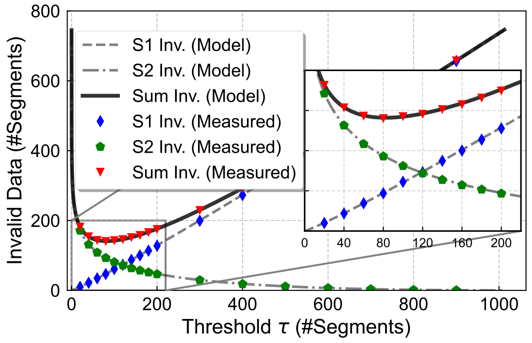


Figure 5: Model validation for different thresholds τ under the two-stream separation setting ($WAF_{target} = 1.2$).

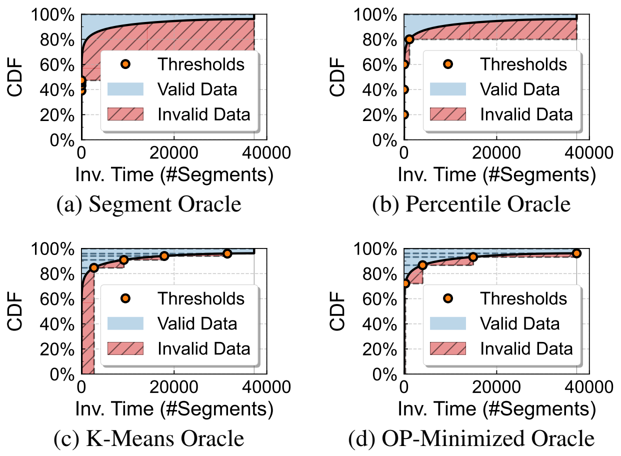


Figure 6: Separation thresholds for different Oracle methods under Zipf 1.01 workload. The red shaded area represents the required OP space to achieve $WAF = 1$.

First, any stream can achieve a WAF of 1 (i.e., zero-cost GC) if it is provisioned with enough OP to accommodate the maximum invalidation time of its data, allowing all data to expire naturally. Second, and more critically, the separation thresholds can be tuned to minimize the total integrated volume of this buffered invalid data across all streams—that is, the total required OP space.

3.3 Theoretical Model: OP-Minimized Oracle

We introduce the *OP-Minimized Oracle*, a theoretical model that optimally determines separation thresholds for a given number of physical streams by formally minimizing the OP space required to achieve a WAF of 1.

As detailed in Algorithm 1, the Oracle first models the required OP space as an objective function, `CalOP`, which takes the workload's CDF and the $K-1$ separation thresholds for K streams as input. It then performs an iterative search (Lines 9-17) to find the set of thresholds that minimizes this function. The `CalOP` function is significant because it calculates the total area under the workload's CDF, partitioned by the given separation thresholds. This area precisely models the required OP space to guarantee zero-cost GC.

Algorithm 1: Finding Optimal Thresholds for OP-Minimized Oracle

```
   Function CalOP(τ, F)
       // Calculates total OP for WAF=1
1      $OP_{total} \leftarrow 0$;
2      for $i = 1 \rightarrow K$ do
           // OP for a stream is its invalid
              data volume
3          $OP_{stream} \leftarrow \int_{\tau_{i-1}}^{\tau_i} (F(t) - F(\tau_{i-1}))dt$;
4          $OP_{total} \leftarrow OP_{total} + OP_{stream}$;
5      return $OP_{total}$;

   Input: The workload's invalidation time CDF $F(t)$,
          number of streams $K$.
   Output: A vector of $K-1$ thresholds $(\tau_1, \ldots, \tau_{k-1})$
           that minimizes OP.

6  $\tau = (\tau_0, \tau_1, \ldots, \tau_{K-1}, \tau_k)$ such that $F(\tau_i) = i/K$;
7  $OP_{min} \leftarrow$ CalOP$(\tau, F)$;
8  $converged \leftarrow$ false;
9  while not $converged$ do
10     $converged \leftarrow$ true;
11     for $i = 1 \rightarrow K-1$ do
           // Find the best value for $\tau_i$ while
              keeping others fixed
12         $\tau'_i \leftarrow \arg\min_{t \in (\tau_{i-1}, \tau_{i+1})}$ CalOP$(\tau|_{\tau_i = t}, F)$;
13         $OP_{new} \leftarrow$ CalOP$(\tau|_{\tau_i = \tau'_i}, F)$;
14         if $OP_{new} < OP_{min}$ then
15             $\tau_i \leftarrow \tau'_i$ // Update the threshold
16             $OP_{min} \leftarrow OP_{new}$;
17             $converged \leftarrow$ false;
18 return $(\tau_1, \ldots, \tau_{k-1})$;
```

This formal optimization stands in stark contrast to prior Oracle-based studies, which rely on ad-hoc heuristics. We compare our work against three such policies: **Segment Oracle** [49], **Percentile Oracle** [7], and **K-Means Oracle** [38]. Figure 6 visually demonstrates the flaws of these heuristics. Both Segment and Percentile Oracles select thresholds that are agnostic to the CDF's shape. While K-Means Oracle uses the actual distribution of invalidation times, its objective is to minimize the within-cluster sum of squares [15]. This objective is *not equivalent to* our goal of minimizing OP space. In contrast, our Oracle's thresholds are placed at optimal cutting points derived from the CDF's shape to formally minimize the total amount of invalid data (the red shaded region).

We empirically validated our approach by comparing the OP-Minimized Oracle against the three heuristic-based Oracles on one synthetic and two real-world cloud workloads [28, 61]. Following prior work, we used a six-stream configuration (5 for user and 1 for GC) and a CB victim

Table 1: Comparison of WAF for different Oracle policies across three workloads and various OP Ratios.

Policy	Zipf-1.01					Backend-Tencent [61]					Backend-Ali [28]				
	OP Ratio (%)					OP Ratio (%)					OP Ratio (%)				
	15	20	25	30	35	15	20	25	30	35	15	20	25	30	35
Segment Oracle [49]	1.366	1.290	1.235	1.195	1.162	1.195	1.162	1.156	1.136	1.119	1.476	1.387	1.291	1.232	1.185
Percentile Oracle [7]	1.231	1.167	1.122	1.090	1.065	1.090	1.065	1.105	1.079	1.063	1.269	1.190	1.136	1.100	1.071
K-Means Oracle [38]	1.081	1.059	1.043	1.030	1.019	1.030	1.019	1.028	1.022	1.017	1.096	1.062	1.038	1.020	1.008
OP-Minimized Oracle	**1.034**	**1.017**	**1.007**	**1.002**	**1.000**	**1.002**	**1.000**	**1.000**	**1.000**	**1.000**	**1.058**	**1.023**	**1.005**	**1.000**	**1.000**

selection policy [49]. The results in Table 1 show that our Oracle fundamentally improves the trade-off between OP space and WAF. At any given OP ratio, our Oracle achieves a substantially lower WAF, reducing the GC writes ($WAF - 1$) by 39.6% to 100% compared to the next-best heuristic. For example, on the Zipf workload, our Oracle achieves a WAF of 1.017 with 20% OP. By contrast, the next-best K-Means Oracle needs 35% OP to achieve a worse WAF of 1.019, an OP ratio at which our Oracle already achieves a perfect WAF of 1.0. This demonstrates that a formal model is essential for unlocking the true potential of data separation.

3.4 The Gap Between Theory and Practice

While the OP-Minimized Oracle establishes a new theoretical limit, its practical implementation is hindered by three fundamental challenges:

Prediction Error. The assumption of perfect foresight is unrealistic; any online predictor is inherently imperfect. While prior work has focused on improving prediction accuracy [38, 47], 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. The Oracle requires *a priori* knowledge of the entire, static invalidation time distribution to find optimal thresholds, which is untenable for online systems. Prior online approaches have resorted to suboptimal compromises: static, offline profiling that cannot adapt to workload changes [7]; simplistic heuristics that fail on complex distributions [47,49]; or complex, model-driven configurations that are brittle to workload shifts, forcing a fallback to simpler, less optimal policies when their models prove inaccurate [38]. The challenge, therefore, is to make near-optimal 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) [7,47].

These challenges motivate the design of MINOS, a practical technique designed to bridge this gap between our theoretical model and practical constraints. We detail its design in the next section.

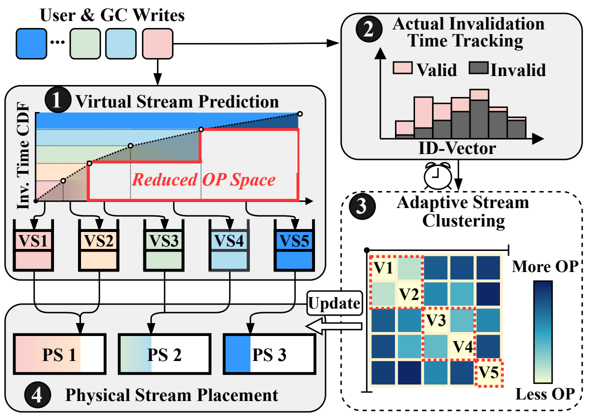


Figure 7: Overview of MINOS.

4 MINOS Design

The design of MINOS is guided by a core principle: to operationalize the optimization strategy of our theoretical Oracle in a practical, online environment. Because searching for these continuous separation thresholds online is intractable, MINOS transforms this continuous search into a tractable discrete problem through a two-level stream architecture. As shown in Figure 7, its workflow consists of four key stages:

❶ ***Virtual stream prediction*** (§ 4.1). MINOS classifies all incoming writes (from both user and GC) into fine-grained virtual streams based on their predicted invalidation time.

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.

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.

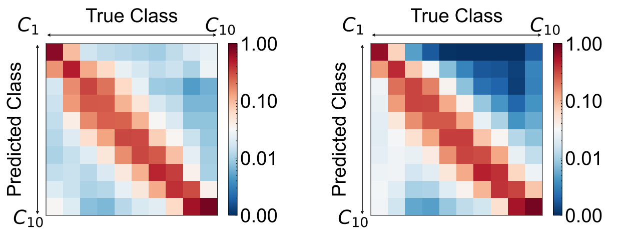
Algorithm 2: Virtual Stream Prediction

```
Input: Metadata $B$, current time $t_{curr}$, I/O type $isUser$
Output: Assigned virtual stream ID $id_{vs}$
1  $\Delta t \leftarrow t_{curr} - B.lastUserWriteTime$;
2  if $isUser$ then
3      if $B.lastUserWriteTime = INITIAL\_VALUE$ then
4          $id_{vs} \leftarrow N_{user} - 1$;
           // Assign new writes to the last
              user stream
5      else
6          Update($userCDF, \Delta t$);
7          $id_{target} \leftarrow$Query($userCDF, \Delta t$);
8          $id_{vs} \leftarrow B.lastUserVsId$;
9          $id_{vs} \leftarrow id_{vs}-$sign($id_{vs} - id_{target}$);
           // Gradually adjust ID by one step
       // Update block metadata
10     $B.lastUserWriteTime \leftarrow t_{curr}$;
11     $B.lastUserVsId \leftarrow id_{vs}$;
12 else
       // Place GC writes based on survival
          time $\Delta t$
13     Update($gcCDF, \Delta t$);
14     $id_{vs} \leftarrow N_{user}+$Query($gcCDF, \Delta t$);
15     $B.lastGCWriteTime \leftarrow t_{curr}$;
16     $B.lastGCVsId \leftarrow id_{vs}$;
17 return $id_{vs}$;
```

4.1 Virtual Stream Prediction

MINOS's first stage discretizes the invalidation time distribution. It transforms the regression problem of predicting invalidation time into a more tractable classification task: assigning data blocks to a virtual stream. To do this, MINOS partitions the observed invalidation time CDF into uniform probability quantiles, with each quantile defining a virtual stream. This quantile-based partitioning ensures that each virtual stream receives a balanced amount of traffic, creating a stable input for the subsequent clustering stage. We maintain these quantiles online efficiently using the P^2 algorithm [19], a lightweight estimator with $O(1)$ time and space complexity.

Algorithm 2 details the prediction logic. While we use a lightweight and common feature, a block's last user write time [38, 47, 49], our design introduces three key refinements for accuracy and stability. First, new user writes are assigned to a dedicated stream ($N_{user} - 1$), as their invalidation behavior is initially unknown (Line 4). Second, for data updates ($[0, N_{user} - 2]$), we employ a novel *correlative prediction* heuristic. Instead of reassigning a block directly to the target stream suggested by its last invalidation time, our logic adjusts its current virtual stream ID by at most one step toward the target (Lines 6-9). As shown in Figure 8, this method produces a deliberately asymmetric error, significantly reducing the costly misclassifications of cold data as hot. Third, writes



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

Figure 8: Confusion matrices for prediction accuracy on the Tencent cloud workload, where invalidation times are uniformly discretized into 10 classes (C_1 to C_{10}).

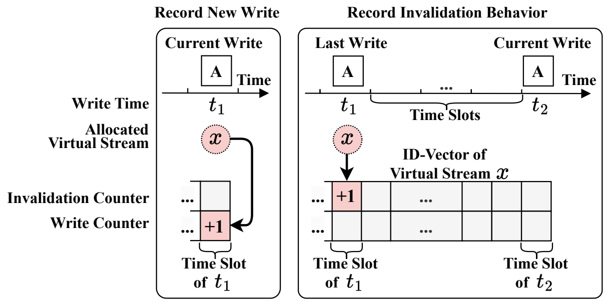


Figure 9: ID-Vector records the *write counter* and *invalidation counter* of virtual streams.

from GC are handled by a separate estimator and mapped to a dedicated set of virtual streams ($[N_{user}, N_{user} + N_{gc} - 1]$), thereby isolating GC writes from user writes (Lines 13-16).

4.2 Actual Invalidation Time Tracking

Any online predictor is inherently imperfect. We argue that the critical challenge is not to endlessly pursue prediction accuracy with more powerful and costly models [7, 47], 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 virtual stream 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 9 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 [38]. 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

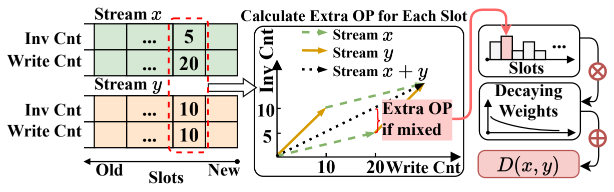


Figure 10: MINOS quantifies the distance between virtual streams as the additional OP required.

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

This stage is the core of MINOS 's online adaptivity. It periodically clusters the fine-grained virtual streams into limited system level physical streams, effectively re-calculating the near-optimal separation strategy for the current workload.

OP-Minimizing Distance Metric. The cornerstone of our clustering algorithm is a distance metric that quantifies the projected cost of merging two virtual streams, x and y. This cost is defined as the *increase in OP space* for a stream resulting from the merge.

To formalize this (detailed in Figure 10), we use the write counter (W_x^t, W_y^t) and invalidation counter (I_x^t, I_y^t) from the ID-Vectors for each time slot t. When merged, streams x and y form a new logical stream with a combined invalidation ratio of $(I_x^t + I_y^t)/(W_x^t + W_y^t)$. The new expected OP space from the blocks originally written by stream x at t becomes:

$$I_{new}^t = \left(\frac{I_x^t + I_y^t}{W_x^t + W_y^t} \right) \cdot W_x^t. \qquad (1)$$

The per-slot distance, $D_t(x,y)$, is the absolute difference between this new expected OP (I_{new}^t) and the original OP (I_x^t). This represents the marginal OP increase for stream x's data, which simplifies to:

$$D_t(x,y) = |I_{new}^t - I_x^t| = \frac{|W_x^t I_y^t - W_y^t I_x^t|}{W_x^t + W_y^t}. \qquad (2)$$

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:

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

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.

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 $M$;
```

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 recalculated. 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

A practical data separation technique must impose minimal overhead. This section details how MINOS reduces its memory and computational overhead and discusses the selection of its key parameters.

Memory Overhead Mitigation. MINOS maintains per-block metadata, including write slots (32 bits) and virtual stream IDs (4 bits) for both user and GC writes. A naive implementation would require 9 bytes per 4KB block, a 2.25× increase over a standard 4-byte mapping pointer [62].

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 [38, 59]. 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 [47] or designs requiring a background core [38], 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 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 [38], evaluates the WAF of different data separation policies across a wide range of workloads and configurations. For system-level evaluation, we implement MINOS and all baselines in CSAL [62], a modern log-structured block storage system built on SPDK [58]. 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 [49].

Baselines. We compare MINOS against a spectrum of policies, ranging from low-overhead heuristics to the SOTA.

- **SepGC** [48]: A minimal-overhead policy that isolates user writes from GC writes.

Table 2: Evaluated workloads.

Workload	Time	#Volume	Working set size (GiB)	Write traffic (TiB)	Average write size (KiB)
Vol-Tencent [61]	2018	62	15425	148	71.4
Vol-Ali [28]	2020	105	18459	357	52.6
BE-Tencent [61]	2018	-	1022	27.7	34.5
BE-Ali [28]	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

- **MiDA** [41]: An extension of SepGC that further separates GC writes based on data migration counts.
- **SepBIT** [49]: 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** [38]: 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, 47].

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 [61] and Alibaba Cloud [28], 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 [52].
- **Synthetic Traces:** We use FIO [4] to generate two Zipf workloads: one hot (Zipf-H, $\theta = 1.01$) and one cool (Zipf-C, $\theta = 0.8$).

5.2 WAF Reduction and Policy Analysis

Overall Performance. Figure 11 presents the average WAF of MINOS and baseline policies across seven workloads under our default configuration. The results demonstrate that MINOS consistently outperforms all baselines. The performance

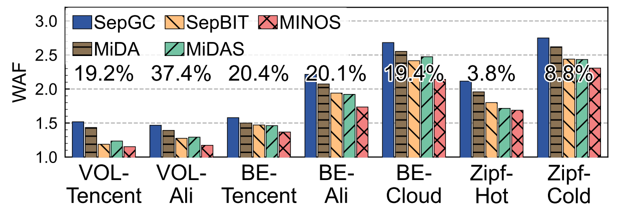
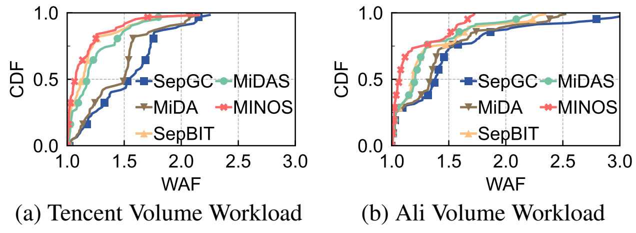


Figure 11: WAF under default configuration.



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

Figure 12: WAF CDF under different volume workloads.

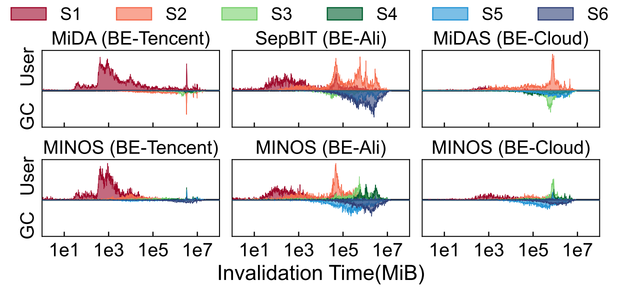


Figure 13: Invalidation time distribution across various streams under the backend workloads.

gains are most pronounced on the five complex, real-world cloud workloads, where MINOS reduces GC writes by 19.2% to 37.4% compared to the next-best policy. On the Zipf workloads, MINOS also delivers consistent improvements, albeit with more modest gains ranging from 3.8% to 8.8%.

Performance of Volume Workloads. To provide a more granular view, Figure 12 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 43.5% 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 20% of these volumes, MINOS dramatically improves performance for the remainder, reducing the median GC writes by 65.6% relative to SepBIT.

Analysis of Data Separation Efficiency. MINOS's superior performance stems from its more effective data separation, as illustrated in Figure 13. 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.

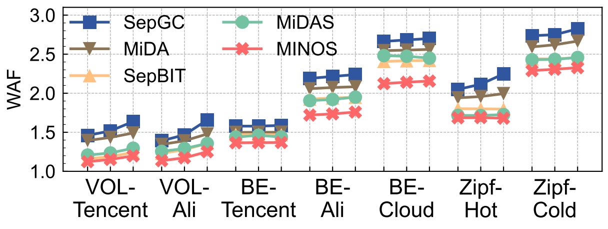


Figure 14: WAF sensitivity to segment size. Within each workload group, points from left to right represent sizes of 256, 512, and 1024 MiB.

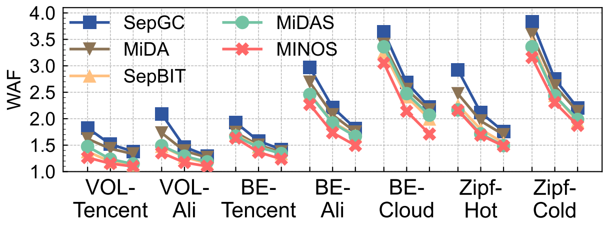


Figure 15: WAF sensitivity to the OP ratio. Within each workload group, points from left to right represent OP ratios of 10%, 15%, and 20%.

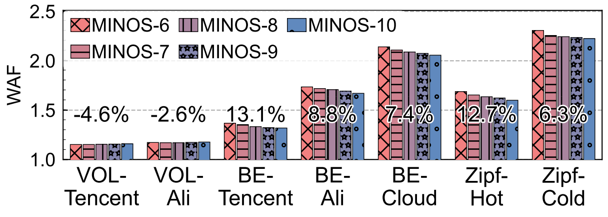


Figure 16: Impact of the number of streams in the system.

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 14. 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 20.6% at 1 GiB to 24.2% at 256 MiB.

Impact of OP Ratio. MINOS's performance advantage grows with the OP ratio. As shown in Figure 15, raising the OP from 10% to 20% increases its average reduction in GC writes on real-world traces from 16.8% to 28.3%.

Impact of Number of Physical Streams. MINOS effectively leverages additional physical streams, demonstrating the scalability of its adaptive clustering. As shown in Figure 16, increasing the stream count from six to ten (the maximum number MiDAS required for the Zipf-C workload) achieves a finer-grained data separation, further reducing average GC writes by 9.7% on large-scale backend and synthetic workloads compared to the default six-stream configuration.

Ablation Study and Algorithmic Parameter Sensitivity. Figure 17 evaluates the core components and parameter sensitivity of MINOS in terms of the average GC write reduction.

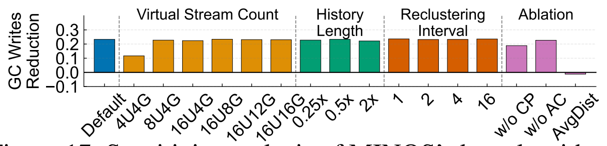


Figure 17: Sensitivity analysis of MINOS's key algorithmic parameters on GC writes Reduction.

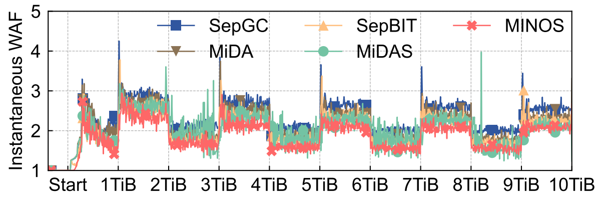


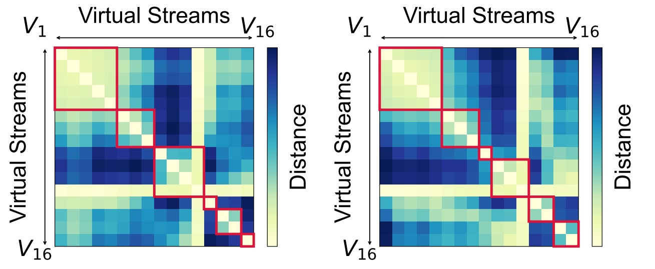
Figure 18: Instantaneous WAF under a dynamic workload that alternates between Zipf-H and Zipf-C every 1 TiB.

Our ablation study first validates three key components of our design. Disabling correlative prediction (w/o CP) significantly degrades performance, confirming that this heuristic is crucial for preventing the costly misclassification of cold data as hot. Furthermore, removing our adjacent clustering heuristic (w/o AC), and instead performing a full search yielded no discernible performance improvement, validating it as a highly effective efficiency heuristic. Finally, to validate our distance metric, we replaced it with a simpler metric based on average invalidation time (AvgDist), a heuristic similar to that of vStream [59]. This change results in a significant performance degradation, confirming the superiority of our formally-guided metric.

Our default parameter settings are derived from this sensitivity analysis. We select a virtual stream count of 12 for user data and 4 for GC data ($N_{user} = 12$, $N_{gc} = 4$) because further increases yield diminishing returns. We use an ID-Vector history length of L equal to the number of segments in the system, as longer histories were found to hurt performance by reacting too slowly to workload changes. Finally, we set the reclustering interval $W_{recluster}$ to 8 segment-writes, which provides a good balance between responsiveness and stability.

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 18, MINOS consistently maintains the lowest and most stable instantaneous WAF, demonstrating its rapid adaptability. This stability is a direct result of MINOS's adaptive clustering, which Figure 19 visualizes in action. At 7.5 TiB, under the cool Zipf-C workload, the system experiences high GC pressure; MINOS responds by allocating three physical streams to separate the internal GC traffic. In contrast, at 8.5 TiB, under the hot Zipf-H workload, GC pressure lessens, and the first two GC virtual streams become statistically similar. MINOS again adapts by merging them and reallocating the freed stream to user writes.

MiDAS, in contrast, exhibits significant WAF volatility. This behavior stems from a tight coupling between its static, model-derived configuration and its victim selection policy. For instance, after the Zipf-C phase (1-2 TiB), MiDAS establishes a 9-stream configuration for the Zipf-C workload. However, when the workload shifts to Zipf-H, this configuration becomes stale. Because the new hot-data groups have not yet reached their pre-configured size, the outdated model is forced to adhere to its static policy: cleaning the cold groups. Reclaiming these low-utility segments directly triggers the severe WAF spikes, a situation that persists until a slow failback mechanism is engaged.



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

Figure 19: Clustering results of MINOS.

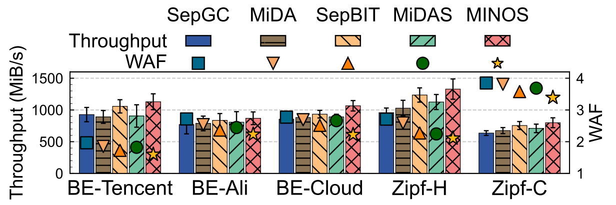


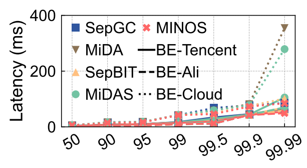
Figure 20: WAF and throughput for CSAL [62] under different workloads and data separation policies.

5.3 System Performance

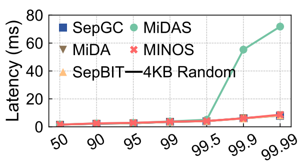
We now evaluate the end-to-end performance of MINOS in our CSAL implementation to demonstrate how its WAF reduction translates into system-level benefits. All experiments run in a single-threaded polling mode with a dedicated, fully-saturated CPU core to handle all I/O and data separation logic. All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

Throughput and WAF. Figure 20 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 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.



(a) Backend workloads.



(b) 4KB random write.

Figure 21: Latency distribution for different workloads.

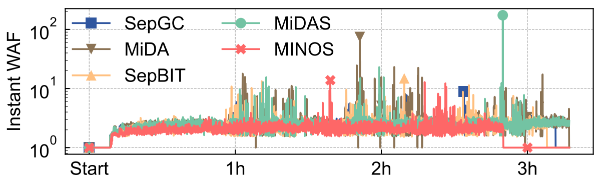


Figure 22: Instantaneous WAF of different data separation policies over time under the BE-Cloud workload.

Latency and QoS. Figure 21a presents the write latency for the backend workloads. MINOS maintains excellent latency and QoS, with a profile nearly identical to the best-performing lightweight baselines. 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 22) caused by 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 other lightweight baselines. In contrast, MiDAS's throughput drops by 6.2% to 76.6K IOPS. As corroborated by Figure 21b, 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, 62]. 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.

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	1.510
MINOS (Agg8, Default)	288	1.514
MINOS (Agg32)	72	1.526
SepGC	∼0	1.892
MiDA	∼0	1.790
SepBIT	590*	1.658
MiDAS	74	1.677

*SepBIT's overhead is based on the realistic snapshot-case analysis from its original paper [49].

Memory Overhead. The effectiveness of our Spatial Metadata Aggregation is detailed in Table 5. Our default configuration (Agg8) imposes a modest 288 MiB memory footprint for a negligible impact on WAF (less than 0.004). This footprint is approximately half of the memory required by SepBIT, which caches metadata only for recent writes via a FIFO queue. For context, MiDAS achieves its 74 MiB footprint by sampling 1% of all data blocks to maintain its Update Interval Distribution and by recording a 2-bit hot filter per block. To create a near iso-memory comparison, our Agg32 configuration nearly matches the memory footprint of MiDAS but still provides a significantly lower WAF, highlighting the fundamental efficiency of our theory-guided method.

6 Conclusion

The performance of modern log-structured storage systems is fundamentally limited by the efficiency of their data separation policies. In this paper, we identified a fundamental limitation in SOTA approaches: even theoretical Oracles with perfect prediction still rely on ad-hoc heuristics to set separation thresholds, leading to suboptimal designs. To address this, we first introduced the OP-Minimized Oracle, which uses a formal objective to find optimal separation thresholds. We then translated this theory into practice with MINOS, a lightweight and adaptive technique that approximates our Oracle online. Our extensive evaluations show that MINOS reduces GC writes by 19.2%–37.4% and boosts system throughput by up to 14.2% over SOTA. By replacing long-standing heuristics with a formal model, MINOS demonstrates a new path toward building significantly more efficient and adaptive data separation techniques.

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 Twentieth 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 Desnoyers. Analytic modeling of ssd write performance. In *Proceedings of the 5th Annual International Systems and Storage Conference*, pages 1–10, 2012.

[13] 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.

[14] 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.

[15] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

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

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

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

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

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

[54] 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.

[55] 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.

[56] 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.

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

[58] 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.

[59] 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.

[60] 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.

[61] 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.

[62] 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.