ZTBP:eBPF-Driven Analysis for Improved Random Read Performance in ZNS Devices via DB Clustering

1st SangHune Jung

Department of Electrical and Electronic Engineering.
University of Yeonsei
Seoul, Rep.South Korea
sanghune.jung@yeosei.ac.kr

2nd Eui-Young Chung

Department of Electrical and Electronic Engineering.
University of Yeonsei
Seoul, Rep.South Korea
eychung@yonsei.ac.kr

***Abstract*—Despite the introduction of advanced storage technologies, NAND-based devices continue to exhibit significantly higher latencies compared to memory technologies like SRAM/DRAM, hindering storage media performance. With the increasing adoption of Zoned Namespace (ZNS) [5] SSD optimizing their performance for random read workloads has a critical challenge to mitigate the inherent latency limitations of NAND devices. This paper presents ZTBP, a novel framework that leverages the eBPF (Extended Berkeley Packet Filter) [10]–[13] technology to trace and analyze NVMe IO operations on ZNS devices, enabling dynamic cache optimization and improving random read performance.**

The ZTBP framework employs eBPF for comprehensive NVMe IO tracing and integrates machine learning libraries for applying DB clustering algorithms to identify critical high-read-intensity zones within the ZNS SSD. This approach facilitates prioritized caching of frequently accessed zones, resulting in enhanced cache hit ratios and reduced random read latencies, addressing the latency bottlenecks in NAND-based storage media.

Extensive evaluations using expected ZNS workloads and demonstrate the effectiveness of ZTBP in optimizing storage performance and mitigating the latency limitations of NAND devices. The proposed framework achieves substantial improvements in IOPS, bandwidth, and cache hit rates compared to baseline scenarios, with performance gains ranging from 14% to 40% across various workload patterns and intensities.

This research introduces a novel approach to address the challenges of random read performance and latency bottlenecks in NAND-based ZNS SSDs by leveraging eBPF for IO tracing and machine learning for cache optimization. The ZTBP framework contributes to the advancement of storage technologies, enabling more efficient and high-performance storage solutions that overcome the inherent limitations of current NAND devices. This study significantly diverges from previous works by specifically targeting NVMe IO operations analysis for performance enhancement, which wasn't the primary focus of the earlier studies on optimizing RocksDB [7] for ZNS SSDs, the benefits of ZNS interfaces [8], or general performance characteristics of ZNS SSDs. [9]

***Index Terms*—eBPF,ZNS,DB Clustering**

I. Introduction

This study introduces a method using eBPF for analyzing NVMe IO operations on ZNS NVMe devices, focusing on DB-clustering. While current research on eBPF mainly addresses NVMe IO tracking and command frequency reduction, its use in reprocessing tasks remains underexplored.

ZNS NVMe devices offer efficient flash device management through direct zone mapping, aligning logical and physical addresses unlike legacy devices. By leveraging these features and statistical machine learning, our research aims to trace and analyze NVMe IO operations comprehensively.

In this paper, we propose it 'ZTBP'

- We hypothesize that despite randomness, certain zones in ZNS devices, each with unique read intensities, are more actively involved in operations. We believe machine learning, especially techniques like KNearest Neighbors (KNN) or DB clustering, can identify these critical zones. Using eBPF for IO activity tracing and categorization, we seek to enable effective predictions.
- eBPF's kernel development bypass and compatibility with traditional machine learning techniques broaden its application, promising insights for storage performance optimization and contributing significantly to the field.

Our system is anticipated to leverage a new tool called eBPF to contribute to performance improvements in storage systems. This approach is expected to make a tangible impact on enhancing the performance of massive server system or big data products using storage solutions.

A. Motivation

The claim that Intel Optane's latency surpasses host IO has sparked discussions, depending on block size and settings. Optane's expected impact hasn't fully materialized [3], with NAND's latency still much higher than SRAM/DRAM Fig. 1. Additionally, enhancing cache hit rates on Host typically necessitated kernel modifications, a complex task due to MMU bugs and the intricate relationship between implementation and storage operations.

ZNS Devices reorganize data into zones, simplifying the logical-physical data relationship and enhancing SSDs' understanding. This zoning is expected to streamline intensive reads

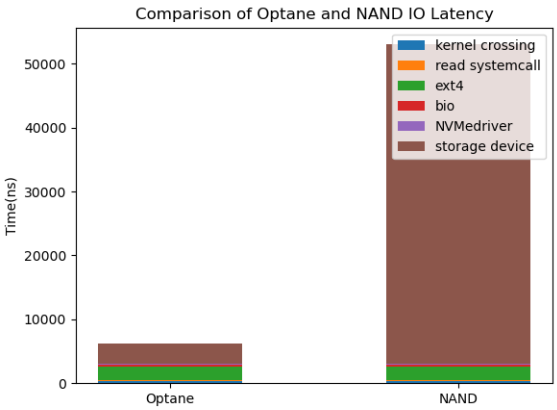

Fig. 1. Comparison of IO Latency for Optane and NAND

TABLE I
COMPARISON OF IO LATENCY IN OPTANE AND NAND STORAGE MEDIA

Storage Media	Optane		NAND	
	Value (ns)	Ratio	Value (ns)	Ratio
Kernel Crossing	351	5.6%	351	0.7%
Read Syscall	199	3.2%	199	0.4%
EXT4	2006	32.0%	2006	3.8%
BIO	379	6.0%	379	0.7%
NVMe Driver	113	1.8%	113	0.2%
Storage Device	3224	51.4%	50000	94.3%

for specific data types, potentially lowering read latency and boosting system performance. Without altering the kernel, the goal is to reduce random-read storage IO latency, integrating well with machine learning libraries for a latency-efficient system. This approach seeks to bypass current cache optimization limits, leveraging ZNS Devices' distinct attributes for a more effective storage solution. [1]

II. DESIGN PRINCIPLES

The design of ZTBP is guided by the following principles:

- **Minimal system resource impact:** The eBPF-based tracing and analysis components are designed to minimize overhead and resource consumption on the host system.
- **Flexibility and adaptability:** The framework allows for easy configuration and fine-tuning of parameters, enabling adaptability to various analysis needs and workload scenarios.
- **Seamless integration with machine learning libraries:** ZTBP seamlessly integrates with popular machine learning libraries, such as PyTorch, TensorFlow, and scikit-learn, facilitating the application of diverse algorithms for data analysis and optimization.

The goal is to develop a solution that bridges the technological gap while minimizing system resource impact and offering flexibility in testing and deployment. Achieving this requires addressing key challenges with strategic solutions, leading to a robust, innovative eBPF-based NVMe IO tracing and analysis framework."Fig. 2"

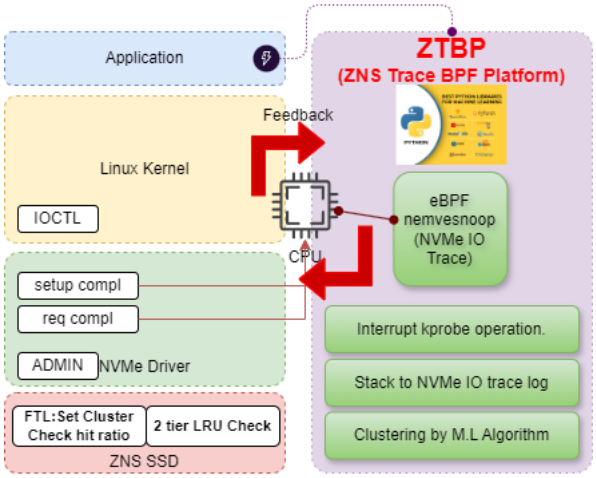

Fig. 2. ZTBP Architecture

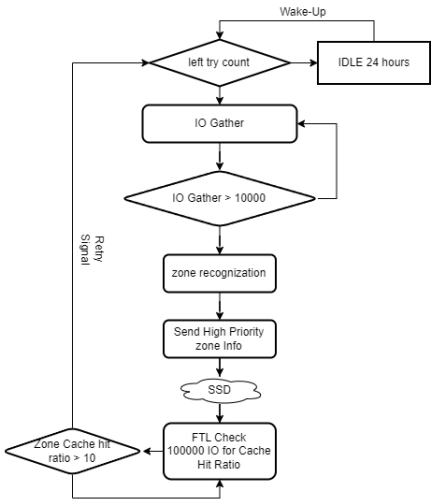

Fig. 3. ZTBP Feedback system for Overhead Optimization

III. IMPLEMENTATION: ZTBP (ZNS TRACE AND ANALYSIS BYPASS KERNEL PLATFORM)

A. Tracing & Gathering Module

*1) **Interrupt kprobe Operation**:* Implements interrupt kprobe operations for real-time event interception within the Linux kernel. Utilizes kprobes to dynamically instrument specific kernel functions, allowing the capture of NVMe IO events as they occur "Algo.1"

*2) **Stacking to NVMe IO Trace**:* Introduces a stacking mechanism to facilitate comprehensive NVMe IO trace collection. Ensures that intricate details of each NVMe IO operation are logged, providing a granular view of the system's behavior

*3) **Machine Learning Library Integration**:* Incorporates a variety of machine learning libraries, including PyTorch, TensorFlow, and scikitlearn. Allows for the application of diverse machine learning algorithms to analyze and interpret NVMe IO traces. "Algo.2"

*4) **Overall Overhead Optimization**:* To reduce host overhead while using eBPF and Python machine learning frameworks, the ZTBP platform is designed to limit its operations to around 10 times per day. Additionally, if a certain level of cache hit ratio increase is not achieved, the overall operation of the platform will be restricted. This approach aims to minimize overhead by carefully controlling the frequency and conditions under which ZTBP operates."Fig.3"

*5) **Pre-Processing for DB Clustering parameter**:* The algorithm performs 5-fold cross-validation to fine-tune DBSCAN's eps and min_samples. It assesses each parameter combination by fitting DBSCAN to training data and measuring clustering quality on test data with the silhouette score, provided multiple clusters exist. After cycling through all folds and parameter sets, it identifies the optimal eps and min_samples that yield the highest average silhouette score, thus optimizing the clustering outcome."Algo.3"

Algorithm 1 Disk Check and BPF Initialization

```
if disk is specified then
    disk_path ← /dev/ + disk
    if disk exists at disk_path then
        Print "no such disk"
        Exit
    end if
    dev ← Get device numbers
    Insert disk filter into BPF text
else
    Remove disk filter from BPF text
end if
if debug or ebpf flag then
    Print BPF text
    if ebpf flag then
        Exit
    end if
end if
Initialize BPF
```

Algorithm 2 Event Processing and Clustering Operation

```
Load disk statistics
Define event processing function
Perform k-fold cross-validation on dataset
Normalize data and apply DBSCAN clustering
while true do
    Poll BPF for events
    if interrupted then
        Exit
    end if
end while
```

B. Feedback Module via IOCTL Command

*1) **Dynamic Interaction by IOCTL command**:* Implements a feedback mechanism through IOCTL commands, fostering dynamic interaction with the ZTBP platform. Users can dynamically control and adjust the behavior of the tracing and gathering module based on realtime requirements.

Algorithm 3 KFold Cross-Validation for DBSCAN Parameter Optimization

```
X ← scale_data
kf ← KFold(n_splits = 5)
best_eps ← None
best_min_samples ← None
best_sil ← −1
eps_values ← range(0.01, 0.05, 0.01)
min_samples_values ← range(20, 50, 10)
progress ← 0
for eps in eps_values do
    for min_samples in min_samples_values do
        for (tr_idx, test_idx) in kf.split(X) do
            X_train, X_test ← X[tr_idx], X[test_idx]
            db ← DBSCAN(eps, min_samples).FIT(X_train)
            test_labels ← db.FIT_PREDICT(X_test)
            if len(unique(test_labels)) > 1 then
                score ←
                sil_score(X_test, test_labels)
                APPEND(sil_avg, score)
            end if
        end for
        if sil_avg then
            sil_avg_score ← mean(sil_avg)
            if sil_avg_score > best_sil then
                best_eps ← eps
                best_min_samples ← min_samples
                best_sil ← sil_avg_score
            end if
        end if
    end for
end for
return best_eps, best_min_samples, best_sil
```

*2) **Fine-Tuning Capability and minimize Total Overhead**:* Enables users to fine-tune parameters and configurations through IOCTL commands, ensuring adaptability to varying analysis needs. Facilitates on-the-fly adjustments with-out the need for platform reinitialization. Such as Clustering information and actual zone hit ratio for retune whole system
Moreover, in order to minize system overhead build a protocol it has try count and check hit ratio for feedback system. which lead to 1 day limitation count and others.

*3) **Zone Recognition index LRU**:* Based on the first index, the cache index structure is con-figured with ZONE recognition offset at insert index 0, and a normal offset in an append manner. In the case of Cache hit operation, the changing index is also based on that index. For normal offset, it cannot exceed the first index, and for ZONE recognition offset, it operates with index 0. In this way, when caching, it is possible to delete only the last index without comparing each cached index if the buffer is full."Fig.4"

Algorithm 4 LRU Cache Operations

```
function CreateNode(key, value, priority)
    allocate newNode
    newNode.key ← key
    newNode.value ← value
    newNode.priority ← priority
    return newNode
end function
function CreateLRUCache(capacity)
    allocate cache
    cache.capacity ← capacity
    return cache
end function
function RemoveNode(node, cache)
    // Adjust links to remove node from cache
end function
function InsertToHead(cache, node, priority)
    if cache is empty then
        if priority is high then
            Set node as both head and tail of cache
        else
            Set node as head, tail, and l_head of cache
        end if
    end if
    if  cache is not empty and l_head exists then
        if priority is high then
            Insert node before current head
        else
            Handle insertion when tail is
            the same as l_head,
            adjusting l_head and tail as needed
        end if
    else cache is not empty but l_head does not exist
        if priority is high then
            Insert node before current head
        else
            Set node as l_head and adjust tail
        end if
    end if
end function
function Get(cache, key)
    // Retrieve node by key and adjust position
end function
function Put(cache, key, value, priority)
    // Insert or update node in cache
end function
function FreeLRUCache(cache)
    // Free all nodes and cache structure
end function
function PrintLRUCache(cache)
    // Print cache details for debugging
end function
```

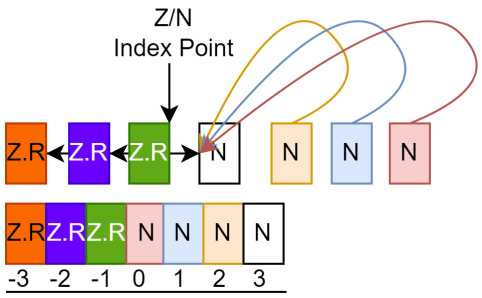

Fig. 4. Zone Recognization LRU.

IV. Evaluations

Introduce the verification protocol designed to validate the ZNS protocol under real workload scenarios and simulate high-bandwidth operations with multi-zone configurations.

A. Verify 'ZTBP' Protocol Test

Execute "NVMeVirt" [6] tests to simulate realworld ZNS workloads. Verify the ZTBP protocol's adherence to specifications and its functionality under diverse workload conditions. Evaluate the system's response to ZNS-specific patterns, emphasizing protocol correctness.

- Environments: QEMU
- Storage: NVMeVirt for ZNS
- BenchMarktool : fio / zone intensive workload
- Cache Size : 64MB
- Zone Size : 32MB

TABLE II
Comparison of System Performance by FIO

Metric	ZTBP Enabled	ZTBP Disabled
Workload Pattern	Zone 1,7,11,15,19 intensive on 80%	
Jobs and Queue	36-Jobs and 4-Queue	
IOPS	79.2k	69.1k
Bandwidth (BW)	278MiB/s (292MB/s)	277MiB/s (290MB/s)
Latency (50th)	1811μs	1876 μs
Cache Hit (%)	30%	26%
Tot Read Cnt	1,000,000	

TABLE III
Comparison of System Performance by FIO

Metric	ZTBP Enabled	ZTBP Disabled
Workload Pattern	Zone 1,7,11,15,19 intensive on 60%	
Jobs and Queue	48-Jobs and 4-Queue	
IOPS	115k	105k
Bandwidth (BW)	422MiB/s (442MB/s)	409MiB/s (429MB/s)
Latency (50th)	1663μs	1876 μs
Cache Hit (%)	24%	17%
Tot Read Cnt	1,000,000	

The ZTBP protocol enhances storage performance and efficiency, significantly increasing IOPS and bandwidth, thus improving data storage and retrieval. These enhancements are

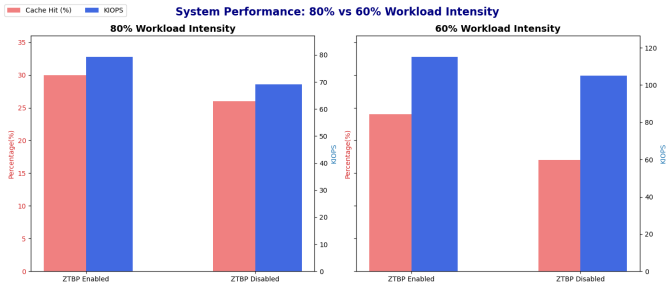

Fig. 5. ZTBP Result 80% and 60% Intensive

crucial for high-throughput, low-latency applications, underscoring the importance of protocol advancements in storage technology evolution.

*1) **Evaluation summary**:* Summarize findings from both verification and simulation-based tests, The verification tests conducted to evaluate the ZTBP protocol under simulated workload scenarios reveal its effectiveness in enhancing storage operation performance. Key findings from these tests include:

- **Increase in Cache Hits** : The application of the ZTBP protocol led to a substantial increase in minimum cache hits, with improvements ranging from 14% to an impressive 40%.
- **Improvement in Maximum Performance** : There was a notable improvement in maximum performance, with gains exceeding 11%.

The ZTBP protocol significantly boosts storage efficiency and performance, especially in high-throughput, low-latency settings. Its impact on cache hit rates and performance improvements demonstrates its potential to advance storage technologies, making it essential for high-performance storage needs.

V. Discussion and Limitation

A. Reliability of the Used ZNS Workload

The study struggles with precise ZNS workload simulations due to data scarcity. Using ZNS's zone-based operations, it aims to predict heavy use in specific zones for realistic workload modeling. Future research will explore zone traits more deeply to improve insights and reliability, tackling bandwidth measurement and variable manipulation challenges.

B. Limitations in Measuring Higher Bandwidth and Manipulating Other Factors

An overarching challenge stems from the constraint posed by the absence of tangible hardware during the developmental and evaluative phases. This constraint severely impedes the capacity to accurately gauge scenarios involving heightened bandwidth and to manipulate various determinants influencing NVMe IO operations. Addressing this predicament necessitates the integration of strategic measures within the design framework, potentially entailing the establishment of authentic simulation environments or the utilization of surrogate apparatus. Surmounting this obstacle holds paramount significance in ensuring the adaptability and efficacy of the eBPF-based NVMe IO tracing and data collection module across a spectrum of operational contexts. To enhance precision in estimating and evaluating scenarios involving elevated bandwidth, the integration of event-based simulation with meticulous timing becomes imperative.

VI. Conclusion

Notably, it enhances ZNS random read performance without the kernel modifications. The traced logs, analyzed in real-time using mainstream machine learning algorithms, enable the dynamic elevation of cache hit ratios. Analyzed ZNS address, the SSD FTL cache hit ratio increases, resulting in improved random read performance. This observation suggests a positive correlation between cache optimization and performance enhancement.
The verification spans from a minimum of 60% to a maximum of 80% workload, affirming the efficacy of ZTBP in optimizing storage performance.

As we look ahead, this research provides a solid foundation for future innovations in storage optimization. The validation process with ZTBP to elevate performance and reduce bandwidth requirements in zone-based workloads. In essence, BPF stands as a promising tool for redefining the landscape of storage device optimization and performance enhancement.
In future research, we will utilize a simulator to optimize host behavior and other factors, systematically approaching how ZTBP operations will precisely affect devices.

References

[1] Zhong, Y., Li, H., Wu, Y. J., Zarkadas, I., Tao, J., Mesterhazy, E., ... & Cidon, A. (2022). XRP:In-Kernel Storage Functions with eBPF. In 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22) (pp. 375-393).

[2] Tehrany, N., & Trivedi, A. (2022). Understanding NVMe Zoned Namespace (ZNS) Flash SSD Storage Devices. arXiv preprint arXiv:2206.01547.

[3] Tom Coughlin.Is Intel Going To Drop Optane?. https://www.forbes.com/sites/tomcoughlin/2022/02/28/is-intel-going-to-drop-optane/?sh=511f22ec79f2.

[4] Zhong, Y., Wang, H., Wu, Y. J., Cidon, A., Stutsman, R., Tai, A., & Yang, J. (2021, June). BPF for storage: an exokernel-inspired approach. In Proceedings of the Workshop on Hot Topics in Operating Systems (pp. 128-135).

[5] Han, K., Gwak, H., Shin, D., & Hwang, J. (2021). ZNS+: Advanced zoned namespace interface for supporting in-storage zone compaction. In 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21) (pp. 147-162).

[6] Kim, S. H., Shim, J., Lee, E., Jeong, S., Kang, I., & Kim, J. S. (2023). NVMeVirt: A Versatile Software-defined Virtual NVMe Device. In 21st USENIX Conference on File and Storage Technologies (FAST 23) (pp. 379-394).

[7] Im, M., Kang, K., & Yeom, H. (2022, November). Accelerating RocksDB for small-zone ZNS SSDs by parallel I/O mechanism. In Proceedings of the 23rd International Middleware Conference Industrial Track (pp. 15-21).

[8] Bjørling, M., Aghayev, A., Holmberg, H., Ramesh, A., Le Moal, D., Ganger, G. R., & Amvrosiadis, G. (2021). ZNS: Avoiding the block interface tax for flash-based SSDs. In 2021 USENIX Annual Technical Conference (USENIX ATC 21) (pp. 689-703).

[9] Shin, H., Oh, M., Choi, G., & Choi, J. (2020, August). Exploring performance characteristics of ZNS SSDs: Observation and implication. In 2020 9th Non-Volatile Memory Systems and Applications Symposium (NVMSA) (pp. 1-5). IEEE.
[10] bcc. [Online]. Available: https://github.com/iovisor/bcc.
[11] bpf: Introduce function-by-function verification. [Online]. Available: https://lore.kernel.org/bpf/20200109063745.3154913-4-ast@kernel.org/.
[12] bpftrace. [Online]. Available: https://github.com/iovisor/bpftrace.
[13] Cilium. [Online]. Available: https://github.com/cilium/cilium.
[14] Cloudflare architecture and how BPF eats the world. [Online]. Available: https://blog.cloudflare.com/cloudflare-architecture-and-how-bpf-eats-the-world/.
[15] DPDK Data Plane Development Kit. [Online]. Available: https://www.dpdk.org/.
[16] eBPF. [Online]. Available: https://ebpf.io/.
[17] Efficient io with io_uring. [Online]. Available: https://kernel.dk/io_uring.pdf.
[18] NVMe base specification. [Online]. Available: https://nvmexpress.org/wp-content/uploads/NVM-Express-1_4b-2020.09.21-Ratified.pdf.
[19] Optimizing Software for the Next Gen Intel Optane SSD P5800X. [Online]. Available: https://www.intel.com/content/www/us/en/events/memory-and-storage.html?videoId=6215534787001.
[20] A thorough introduction to eBPF. [Online]. Available: https://lwn.net/Articles/740157/.