PERFORMANCE AND ENDURANCE CONTROL IN EMERGING STORAGE TECHNOLOGIES

A Dissertation
Submitted to
the Temple University Graduate Board

In Partial Fulfillment
of the Requirements for the Degree of
DOCTOR OF PHILOSOPHY

---

by
Tanaya Roy
December 2021

Examining committee members:

Dr Krishna Kant, Dissertation Advisory Chair, Department of Computer and Information Sciences
Dr Chiu Tan, Department of Computer and Information Sciences
Dr Justin Shi, Department of Computer and Information Sciences
Dr Saroj Biswas, Department of Electrical and Computer Engineering

ABSTRACT

The current diverse and wide range of computing moves towards the cloud and demands high performance in low latency and high throughput. Facebook reported that 3.3 billion people monthly and 2.6 billion people daily use their data centers over the network. Many emerging user-facing applications require strict control over the storage latency's tail to provide a quality user experience. The low-latency requirement triggers the ongoing replacement of hard drives (HDDs) by solid-state drives (SSDs) in the enterprise, enabling much higher performance and lower end-to-end storage latencies. It becomes more challenging to ensure low latency while maintaining the device's endurance ratings. We address this challenge in the following ways: 1. Enhance the overall storage system's performance and maintain the SSD endurance using emerging Non-volatile memory (ENVM) technology. 2. Implement deterministic latency in the storage path for latency-sensitive applications. 3. Provide low-latency and differentiated services when write-intensive workloads are present in a shared environment. We have proposed the performance and endurance-centric mechanisms to evaluate the tradeoffs between performance and endurance. In the first approach, our goal is to achieve low storage latency and a long lifetime of the SSD simultaneously, even for a write-heavy workload. Incorporating a significantly smaller amount of ENVM with SSD as a cache helps to achieve the said goal.

SSDs using the NVMe (Non-Volatile Memory Express) interface can achieve low latency as the interface provides several advanced features. The second approach

has explored such features to control the storage tail latency in a distributed environment. The "predictable latency mode (PLM)" advanced feature helps to achieve deterministic storage latency. SSDs need to perform many background management operations to deal with the underlying flash technology traits, the most time-consuming ones being garbage collection and wear leveling. The latency requirement of latency-sensitive applications violates when the I/O requests fall behind such management activities. PLM leverages SSD controllers to perform the background operations during a window, called a "non-deterministic window (NDWin)". Whereas during the "deterministic window (DTWin)", applications will experience no such operations. We have extended this feature in the distributed environment and showed how it helps achieve low storage latency when the proposed "PLM coordinator (PLMC)" is incorporated. In a shared environment with write-intensive workloads present, result in latency peak for Read IO. Moreover, it is required to provide differentiated services with multiple QoS classes present in the workload mixture. We have extended the PLM concept on hybrid storage to realize the deterministic latency for tight tail-controlled applications and assure differentiated services among multiple QoS applications. Since nearly all of the storage access in a data center is over the network, an end-to-end path consists of three components: The host component, Network component, and Storage Component. For latency-sensitive applications, the overall tail latency needs to consider all these components. In a NAS (Network Attached Storage) architecture, it is worth studying the QoS class aware services present at the different components to provide an overall low request-response latency. Therefore, it helps future research to embrace the gaps that have not been considered yet.

DEDICATION

Dedicated to my parents Mrs. Sagarika Roy and Mr. Nirmal Roy, and my elder brother Mr. Subhajit Roy.

ACKNOWLEDGEMENTS

I would like to thank many peoples, who supported my effort on this journey. I want to express sincere gratitude to Temple University and the CIS department for providing me the opportunity to pursue the Ph.D. program. My research advisor, Professor Krishna Kant, has provided immense guidance in this journey. He helped me to gather experience from industry-sponsored projects. This experience is precious to continue the future journey in storage system-based industries and further research work. The other thesis committee members, Professor Justin Y. Shi and Chiu C. Tan, provided their valuable guidance and cooperated in this workflow. I hope that I will apply the learning in this process in the next career step.

I was involved with multiple industry-funded and NSF-funded projects. Tony Floeder from DELL EMC Inc. provided his valuable guideline during the initial days of the Ph.D. program. Working with him was inspiring as well as enjoyable. Dave Minturn from Intel Corp. shared his valuable knowledge and helped to pursue projects in emerging storage technologies and related modern features. Involvement with such great people helped to find my interest in the storage system industry.

Internships at DELL Technologies, Minnesota, and NetApp Inc at Sunnyvale helped engage in modern storage system products involving many people. I would like to thank Jim Rhode and Curt Thiem, my managers from DELL and NetApp respectively, for helping to acknowledge my interest in the industry and start my future journey with NetApp.

The journey would not have been possible if I could not bring the best version of myself to work. My parents and siblings encouraged throughout to pursue the work. The lab mates were great to work together and involved in the collaboration. It was great to meet amazing friends and colleagues to communicate and organize events like Research Mixer. I would like to thank everyone who was always beside me uninterruptedly these years.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION TO THE EMERGING STORAGE TECHNOLOGIES

In this section, we briefly describe the essential characteristics of the emerging storage technologies.

1.0.1 Magnetic Recording Storage Technologies

The most widely deployed storage technology currently is the traditional hard disk drive (HDD); however, this technology is by no means static and is itself undergoing rapid advancements. Much of the action is in advanced magnetic recording techniques which includes many techniques in different stages of development. For example, perpendicular recording (i.e., magnetization perpendicular to the disk surface) is already well established. Other techniques include 3D recording (i.e., being able to select one of several layers and reading/writing data on it), heat assisted magnetic recording (HAMR), microwave assisted magnetic recording (MAMR), heated dot magnetic recording (HDMR), etc. The HAMR technology, already demonstrated to product 16TB drives, heats the point where the magnetic pole is to be flipped to a high but sub-curie temperature to make the flipping easy and then cools it down rapidly. MAMR technology accomplishes similar goal (i.e., reduction of coercivity)

using microwaves. These technologies have and will likely continue to keep HDDs an essential part of the storage hierarchy on the high capacity end, in spite of rapid advancements in solid state drives (SSDs). In fact, variants of the HDD technology such as SMR Feldman and Gibson (2013) (shingled magnetic recording) or IMR Hwang et al. (2017) (interlaced magnetic recording) further increase the track density but at the cost much more difficult random data updates. The idea behind SMR technology is to record successive tracks so closely that while they can be read individually, writing a single track disturbs the nearby tracks. The recording is done in a series of "zones" with either gaps between them (in SMR) or normal spacing zones (in IMR). Thus it is possible to write an entire SMR zone sequentially without losing any data but random writes will not work. Consequently, these technologies are initially targeted towards backups and archiving, but can also find other uses where the IO is largely sequential.

1.0.2 Flash Storage Technology

The increasingly popular storage media is the solid-state drive (SSD) based on the flash technology. It is rapidly augmenting or replacing HDDs Norman (2009); Cobb and Huffman (2012); Cai et al. (2017) in "higher" tiers (i.e., defined as those logically closer to CPU) of the storage hierarchy. The flash technology is based on the "floating gate transistor", i.e., a MOS transistor with an extra gate encapsulated entirely inside the SiO_2 layers but not directly connected to anything. By applying a high positive voltage to the normal gate which sits above the floating gate, electrons can be forced into the floating gate. If the gate voltage is now withdrawn, the electrons get trapped inside floating gate and this forms the nonvolatile memory. The only way to force the electrons out is by applying a large negative voltage to the regular gate, a process known as "erasure". Unfortunately, the erasure is very energy intensive and is designed to be done one entire "block" at a time. The "block" here refers

to flash block and could be rather large – at least 256KB and continuing to go up due to technological considerations and could already be as high as 8MB. A block is divided into "pages" that are 2KB or larger. While the reads and writes are in the units of pages, because of the erasure process, it is not possible to overwrite a page once written. Instead, any intended overwrite must be recorded elsewhere along with suitable metadata to ensure that the latest copy of the page is always read. A written page becomes "clean" (or writeable) only after the entire block containing the page is erased. The erasures are usually done according to some delayed policy (e.g., periodically or when number of clean blocks dips below a threshold) by a process known as *Garbage Collection* (GC). The extent of GC is related to how much extra space is available to manage out-of-place writes (often known as *Space Amplification* or *Overprovisioning*) and how many page writes does the SSD perform internally for single page write issued by the OS (often known as *Write Amplification*). Given the complexity of these management tasks, SSDs have inbuit computer which runs a layer called "File Translation Layer" (FTL) to make the SSD look like a normal HDD.

The trapped electron based operation leads to several other aspects that need to be managed. First, the electrons do tend to leak out eventually, thereby leading to limited "Retention". That is, the number of times a written page can be read continuously is not unlimited. Similarly, an SSD left alone for a very long time may not retain the information, and thus not suitable for long term storage (10's of years or more). Also, when reading or writing a "cell" (or a floating gate transistor), nearby cells may be affected – known as "read disturb" and "write disturb" problems respectively. However, the most important limitation is the "endurance", or the number of times a block can erased (and then reprogrammed). This is commonly specified in terms of PE (program-erase) cycles that a cell can endure before it can no longer be programmed reliably. The standard solution to deal with certain cells being heavily written and thereby wearing out quickly is "wear leveling". Internally,

the FTL implements a virtual to physical address translation at the block level, so that it can transparently ensure that all blocks are programmed roughly at the same rate. Individual blocks could have significant variations in their life-times, and increasingly the wear leveling schemes take this variability into account.

The flash technology started with binary cells, i.e., cells with trapped electrons representing a "1" and one without representing a "0". This is known as SLC (single level cell) recording. It is the simplest, fastest, and can provide about 100K PE cycles of endurance per block, which is adequate in most cases. Later versions of the technology have tried to stuff multiple bits per cell by varying the number of trapped electrons to represent different levels of programming. As an illustration, we could have 4 levels as follows: 0: no electrons trapped, 1: 10 electrons trapped, 2: 20 electrons trapped, and 3: 30 electrons trapped. This then represents 2 bits/cell, incorrectly known as MLC (multi-level cell). MLC can provide about 20K PE cycles. More recently, the fastest growing technologies are TLC (3 bits/cell) with endurance in the range of a few thousand PE cycles, and QLC (quatro level cell or 4 bits/cell) with endurance measured in less than a thousand PE cycles only. The last two are clearly deficient in terms of endurance; however, with increasing size of the drives and a good wear leveling, the deficiency can be made less problematic. For example, a 1TB TLC drive with endurance of 2000 PE cycles, can, in theory, take 1PB of writes before it is exhausted. Given a rated 5 year life-time of the SSD, this amounts to 400 TB/year or 1.08 GB/day. This may be workable for data that's updated only infrequently; however, there is no doubt that the problem of endurance becomes more serious with more bits/cell. Unfortunately, all of the other problems (including retention, read disturb, write disturb, unreliable reads, unreliable writes, etc.) become more pronounced with shrinking feature size of the flash technology (which reduces the overall number of trapped electrons) and with increasing bits/cell.

1.0.3 Other Emerging Storage Technologies

Beyond the Flash technology, numerous nonvolatile memory technologies have been in development, including Phase Change Memory (PCM), Spin-torque transfer RAM (STTRAM), Toggle MRAM (TMRAM), etc. Generally, these technologies are byte addressable (unlike SSD or HDD), are much faster than flash, are re-writable, and have high endurance and retention Mittal and Vetter (2016). Their current downside is the high cost and various technology maturity issues.

The technologies operate using very different principles. For example, the PCM technology works much like writeable CD technology; for writing it quickly melts a phase-change material cell and let's it cool down. A slow cool results in crystalline structure (say a "1") and a fast cool results an amorphous structure (say a "0"). (This actually means that writing a 1 takes longer than writing a 0, which has implications for suitable low-level coding mechanisms). Obviously, this makes writing much slower than reading, but the read speed is only 2-4X slower than DRAM.

Spin-transfer Torque MRAM, or STT-RAM, records data by manipulating the spin of the electrons. Electrons inherently have a small angular momentum (called spin). Electric current is generally unpolarized (consisting of 50% spin-up and 50% spin-down electrons); but by passing the current through a thick magnetic layer (called the "fixed layer"), one can produce a spin-polarized current. If this spin-polarized current is directed into a second, thinner magnetic layer (the "free layer"), the angular momentum can be transferred to this layer, and even flipped. STT-RAM has the potential to simultaneously provide nonvolatility, DRAM-like memory latencies, and extremely high retention and endurance. The technology is still very new but Everspin Electronics has recently released 1Gb chip that operates like a DDR3 memory.

Toggle MRAM is a related technology that uses a magneto-resistive toggle cell.

It uses a transistor coupled with a magnetic tunnel junction (MTJ) memory cell consisting of a fixed magnetic layer, a thin dielectric tunnel barrier and a free magnetic layer. When a bias is applied to the MTJ, electrons that are spin polarized by the magnetic layers tunnel through the dielectric barrier and change the resistance of the MTJ device. The MTJ device has a low resistance when the magnetic moment of the free layer is parallel to the fixed layer and a high resistance when the free layer moment is oriented anti-parallel to the fixed layer moment. During a read, the pass transistor is activated and data is read by comparing the resistance of the cell to a reference device. During writes, the magnetic field from two Write Lines writes the cell at the intersection of the two lines but does not disturb other cells on either line.

One technology that is already available commercially and gaining rapid popularity is the Intel "Optane" Intel (2015, 2018). It is organized as a 3D cross-point structure and believed to be based on the PCM technology, although Intel has not revealed the underlying technology. It is 10X faster and has 30X higher endurance than Flash (QLC SSD), but 20X more expensive; therefore, a suitable use for it in the current storage hierarchy is in form a nonvolatile cache for enhancing both the overall storage performance and endurance (by judiciously deciding whether to accumulate writes in Optane or directly in the SSD).

With DRAM-like latencies and byte-addressability, the emerging technologies are challenging the traditional distinction between memory and storage. The CPU accesses memory in small chunks (i.e., "cachelines", typically 64B in size) and waits until the access is complete. In contrast, the storage access usually is in large chunks (e.g., 4KB or larger) and asynchronous, i.e., the CPU issues the storage access request, and then does something else until informed of the IO completion. Technologies such as STTRAM can also be used as "persistent memory" when used like a memory. In fact, Intel has already released Optane based persistent memory modules that plug into regular DDR3 memory slots. The developments lead to the notion of so called

"Storage Class Memory" (SCM), which provides unified view of memory and storage, and the data using the volatile or nonvolatile storage or different types of nonvolatile storage depending on where it is best to keep the data. Also, whether to wait for a transaction inline or not can decided dynamically based on the latencies, as proposed in J. Gim and Kant (2015). Persistent memory does bring additional challenges precisely due to its data retention capability, since the data in the persistent memory at the point of machine crash may not be consistent or in a known state and thus not usable following a crash. One instead needs a transactional approach to persistent memory use and suitable journaling mechanism to enable restart from a consistent state following the crash.

There are also other storage technologies using new materials or paradigms. For example, Anderson et al. Anderson et al. (2018) discusses using glass for storage, Organick et al. Organick et al. (2018) discusses DNA storage for random data, and Caferty et al. Cafferty et al. (????) discusses storage using small organic molecules. DNA based archival storage is discussed in Bornholt et al. (2016).

1.1 Characteristics of Modern Storage Systems

The most crucial requirement for efficient access to "big data" is the storage system that can manage large volumes of data and distribute them across devices and storage servers for low latency access without stressing individual devices, interfaces or the network. In the following we discuss some of these aspects and point out their role providing low data access latency and high throughput.

1.1.1 Storage Interfacing

Traditional HDD technologies are limited by the device capabilities, and thus the speed of the hardware/software interface between the device (or "target") and the host is not that important. However, with the emergence of SSD and other modern

storage technologies, both the latency and throughput capability of the interface has become crucial. This has led to new developments interfacing technologies as well. In particular, the traditional SATA (Serial ATA) and SAS (Serial Attached SCSI) technologies are rapidly being replaced by the NVMe interface operating directly over the PCI-Express (PCI-E) bus and using its lightweight "queue-pair" mechanisms. NVMe provides not only a low-overhead/low-latency transfer mechanism, but also a number of other features including: (a) Quality of service (QoS) in data transfer with multiple queues along with priority and weighted round-robin disciplines, (b) NVMe sets, that can effectively carve out multiple virtual devices on a single real device, (c) Endurance classes, which can guide out-of-place writes and garbage collection on SSDs. Data transfer to/from a fast NVMe SSD can be done in about 100μs and to/from Optane in under 50μs. The peak data rates supported can be tremendous; with several current SSDs capable of doing sequential reads in excess of 25 Gb/sec. An Optane drive can go even faster.

1.1.2 Storage Networking

Since the data storage needs have always outpaced the storage that can be deployed on one server, much of the storage access in a large system is typically over the local network. This is true regardless of whether the storage is concentrated in a central place or distributed over many servers. Traditionally, with HDD based storage, the network bandwidth required or the network latency incurred in the remote storage access was negligible; therefore, storage was largely concentrated into a set of "storage towers", and accessed over either the regular network (e.g., Ethernet) or specialized network such as Fiber channel. Distributed storage using the object model is becoming quite popular and allows for objects to reside anywhere. However, with SSDs and newer technologies capable of driving tremendous bandwidths and able to provide data with very low latency, both the network bandwidth and latency become

important. Thus, specialized protocols such as NVMe over Fabric (NVMe-OF) are emerging to seamlessly support remote NVMe storage Minturn and Metz (2015); such protocols allow NVMe based access to defined for devices located remotely across the network. However, network issues such as bandwidth allocation, congestion notification/management, and network level QoS become crucial for supporting efficient storage access. In particular, storage access bursts from even a few current SSDs can exceed the bandwidth of a 100 Gb/sec Ethernet link. Thus mechanisms to monitor congestion and take corrective actions become essential.

1.1.3 Storage Virtualization

The need to migrate data across multiple devices or storage servers requires a dissociation between an application's addressing of data from its physical location. For example, the data that is currently hot may be brought in from a lower tier to the highest tier for more efficient access without affecting application's ability to reference it. This requires the application to refer to the data in a device and location independent way, such as a unique data chunk id, which is mapped to a physical address (e.g., a combination of device, volume, and physical chunk number) transparently by the "storage virtualization layer" which is ubiquitous in current large storage systems. Since logical to physical address translation is required for every access, it is usually managed via an appliance that can provide very low lookup latency and high lookup throughput. However, the downside of the appliance is the funneling of all requests through a single node which may be unscalable for large systems. Other techniques are also used, such as built-in translation capability in the local switch that the host connects to, or the host software itself. These all have their pros and cons. For example, switch based translation is the fastest and least likely to face congestion, but such a solution is expensive. Host based virtualization can be slow and inelegant since the host would need to track data mapped to all storage

servers.

CHAPTER 2

INTRODUCING RESEARCH OBJECTIVES AND GOALS

"Performance is king" for applications, such as high-performance computing (HPC), game developing, data-center applications (Facebook, Google, Alibaba etc..), high-frequency trading. The world is world is creating 2.5 quintillions of data everyday and the number is growing 25% every year(Tesfay et al., 2018). This growth of information demand the massive improvements not only on computation but also on storage and network systems. The performance requirement becomes critical when storage devices are network-attached. Therefore, both storage/network infrastructure and associated protocols demand technology changes to manage and store the data. The storage hierarchy evolves and involves solid-state drive (SSD) replacing the traditional hard disk drive (HDD). Moreover, sophisticated storage access protocols are in progress to address the recent trends in user-facing applications. This thesis has addressed some emerging storage technology and low latency guarantee features for further improvement and address the latency-critical application's demand.

2.1 Research Objective And Goals

Storage systems form a crucial component of data center infrastructure and substantially influence the performance of the applications. With applications becoming ever more data centric, this reliance on a fast storage system that can provide the required quality of service (QoS) treatment becomes paramount. Therefore, our objective in this thesis is to propose suitable storage design and IO handling mechanism that can achieve the application's QoS requirements. However, the data-intensive applications place substantial stresses on SSDs, which might affect the device's endurance. Therefore we had three different goals to achieve in our proposed storage model.

Our first goal is to address the endurance limit of an SSD when used for write-intensive workloads. ENVM is proposed to use as cache to enhance the performance. However, what proportion ENVM needs to be used by read and write data to enhance the performance and maintain the endurance of the overall storage hierarchy while reducing the data loss is never studied before. So our goal was to address them and propose a hybrid storage model with SSD paired with ENVM.

For measuring the performance, the average latency is not always suitable. In the case of latency-critical applications, it is essential to provide uninterrupted low latency. Therefore, tail latency plays a vital role in these applications to estimate the worst-case latency experienced. Our next goal was to achieve the deterministic latency for high QoS.

The deterministic latency guaranteeing device features were proposed but not available yet to any commercial SSDs. Our next goal was to emulating a deterministic latency feature in existed storage model.

The application QoS requirement addresses deterministic latency for both storage and network component when storage accesses are performed over the network.

NVMe storage protocol brings low latency access for storage. Moreover, several high-performance fabrics, such as NVMe-OF (NVMe over fabric) (Guz et al., 2018), PM-OF(Persistent memory -over fabric) (Rudoff, 2013; Neal et al., 2020)are in use. Currently, no readily available mechanism coordinates both storage and network. The end-to-end(E2E) latency is defined as a round-trip of request-response pair of an IO request. The E2E can be realized in multiple ways in network-attached storage (NAS) setting, point-to-point, and point-to-multipoint. In point-to-point E2E, the application running on a host requests to a single target. Whereas, in point to multi-point, application requests may spread over multiple storage targets. Therefore, the latter is more complex to handle as it needs to consider all the IO paths between host to multiple targets and the storage technologies present at multiple targets. Currently, we are considering point-to-point E2E latency control in the NAS context. Our goal is to thoroughly investigate the underlying complexity and exploring opportunities to design a suitable solution for remote access storage devices.

2.2 Contribution: Enhancing Endurance of SSD Based High-performance Storage Systems Using Emerging NVM Technologies

SSD technologies with multiple bits per cell are desirable for increasing the performance of data-intensive applications but suffer from poor endurance. Therefore, it is essential to explore how both can be achieved at the same time. Maintaining the endurance rating of a NAND device is crucial, and therefore, the amount of write IO completion needs to be handled cautiously. Pairing the emerging non-volatile memory (ENVM) technologies and SSD can help enhance the storage systems' endurance and performance. While in-memory caching can enhance performance, it increases the potential for data loss. With the high-speed ENVM technologies, it becomes possible to persist writes immediately, thereby eliminating or minimizing data loss, reducing the stress on the back-end SSD, and improving its endurance. Integrated DRAM

and ENVM caches management may help us achieve the desired trade-offs between data loss risk, performance, and endurance. We have proposed performance and endurance-centric mechanism with the paired storage model of SSD and ENVM to explore the trade-offs. With performance as the primary objective, the hybrid storage model's overall read latency can be reduced to 58% of the original by using Intel Optane cache that is only $\sim$ 3% of the SSD size. Similarly, with endurance as the primary objective for both the SSD and Optane storage, the Optane size only needs to be 12% of the SSD size.

2.3 Contribution: PLMC: A Predictable Tail Latency Mode Coordinator For Shared NVMe SSD With Multiple Hosts

Currently, "deterministic" access latency requirements are in a growing number of scenarios. Data sharing has become popular among the different priority applications. However, SSD involves a complex set of management activities in the background, resulting in unpredictable delays and occasional extended access latencies. Therefore, an uninterrupted microsecond-level latency requirement guarantee is essential. The solution becomes even critical when write-intensive workloads are active among the actively shared applications.

To address the ongoing demand of deterministic latency, we have explored *Predictable Latency Mode* (PLM) defined in the NVMe storage access protocol. It is an NVMe feature that enables tightening tail latency in SSDs. We have extended the traditional PLM feature from a single-host environment to multi-host shared settings considering a read-intensive workload mixture of various priorities. We propose a *PLM Coordinator (PLMC)* that regulates access to the PLM of a shared SSD device based on the hosts' traffic characteristics. Our simulation experiments show that the proposed PLMC can achieve 82% improvement in 99.99% tail latency of a bare SSD without PLM. Moreover, the proposed coordinator with simple traffic prediction can

perform 93.2% better than without a coordinator on the 99%-tail latency values.

2.4 Contribution:PLM^{light}: Emulating Predictable Latency Mode In Regular SSDs

The proposed PLM feature is useful but not yet available with any SSD device yet. SSD involves critical management activities. These activities help the device to perform with microsecond-latency most of the time. After experimenting with SSD and from literature, we have realized that SSD can generate millisecond-level latency when critical background operations interfere with the regular IO requests. This long latency varies with SSD technologies. Therefore, a write-intensive workload crowd in a sharing environment needs extra attention to achieve the tail-latency requirement. Therefore, our next attempt was to guarantee low latency in a write-intensive workload mixture.

The modern user-facing application expects steady low latency as the response. The end-to-end latency, defined by the round-trip time of an I/O request, consists of the host, network, and storage components. For profitable user applications, it is crucial to provide differentiated services at each component on the request and response path length. We further explored the possibility of emulating the PLM feature in regular SSDs using Intel Optane, which provides a rather deterministic access latency. In particular, we propose a write I/O friendly PLM^{light}coordinator (PLMLC) that buffers write in Optane and sends them to SSD during the NDWin-like period and intelligently manages the limited number of IOs possible during the DTWin-like period. The coordinator is designed to handle requests from multiple hosts that may access shared data on the SSD and may have different QoS requirements in terms of latencies. The results show that PLMLC improves the 99%-ile tail latency by 5.8x even without sophisticated traffic estimation procedures.

2.5 Outline Of The Dissertation

The outline of this document is as follows. Chapter 2 elaborates the detailed study of write-intensive IO impacts on the SSD's endurance. We have explained that our proposed hybrid storage, SSD paired with ENVM, can help maintain the SSD's endurance ratings besides enhancing the overall storage system performance. Chapter 3 introduces the performance enhancement with deterministic latency using PLM feature in shared environment with read-intensive workload. We have portrayed the architecture of our proposed PLMC with extensive evaluation results. We have showed with simple traffic prediction mechanism PLMC can improve up to 31% in the 99% tail latency of the highest QoS class as compared to an uncoordinated operation of the host. In chapter 4, we have explained the opportunities in E2E QoS latency improvement.

CHAPTER 3

MANAGING TRADE-OFFS BETWEEN PERFORMANCE AND ENDURANCE OF NAND SSDS

3.1 Introduction

A diverse and wide range of scientific computing are moving towards cloud, and demand high-performance both in terms of low latency and high throughput (Shi et al., 2017; Luu et al., 2015). The data intensive computation of these applications places substantial stress on the storage system, which increasingly determine the ultimate performance achieved by these applications.

The primary storage in HPC systems is rapidly evolving from hard disk drive (HDD) to flash technology based solid state storage (SSD) due to their much higher performance and lower energy consumption. However, due to increasing data storage requirements of high-performance computing applications (Liu et al., 2016b, 2012), the Flash technology used by SSDs is moving towards multiple bits/cell to deliver higher storage densities at a lower cost. Unfortunately, as the number of bits/cell increases, the endurance of the device (in terms of number program-erase cycles per cell before the cell wears out) goes down rapidly. In particular, while an enterprise class MLC (multi-level cell, or really 2 bits/cell) can provide 10K PE cycles, the

most recent QLC (quad level cell or 4 bits/cell) provides endurance only in a few hundred PE cycles. Thus while it is attractive to utilize QLC or TLC (triple level cell or 3 bits/cell) in large HPC storage due to their low cost and dramatically higher performance as compared with HDDs, the endurance can become a highly limiting factor for large IO intensive HPC workloads.

Fortunately, the more recent developments in the storage technologies offer a solution that we explore In this work. The space of new nonvolatile memory (NVM) technologies is highly active with a range of technologies, all of which are faster than flash, provide much higher endurance, and quite unlike flash allow overwrites. Some examples of these technologies are Phase Change Memory (PCM), Resistive RAM (RRAM), Spin-torque transfer RAM (STTRAM), 3D Xpoint etc. We henceforth refer to them as "emerging" NVM or ENVM technologies. One such ENVM technology has recently been made available commercially by Intel under the name "Optane", organized as a 3D cross-point structure. It is 10X faster and has 30X higher endurance than Flash (QLC SSD). Currently, it is 20X more expensive; therefore, a suitable use for it in the current storage hierarchy is in form of a nonvolatile cache. Accordingly, we will analyze 3-layer storage hierarchies In this work consisting of DRAM, Optane, and SSD layers, and examine how a small amount of Optane can provide both high-performance and high overall endurance for the HPC storage hierarchy.

In addition to performance, an important concern in HPC systems is that of resilience, and the resilience issues become more severe as the size of the system increases. One important aspect of resilience is the time it takes to recover from machine crashes. The amount of unsaved data lost is an important aspect of recovery time. With the availability of high speed NVM technologies such as Optane, it is possible to minimize the data loss by minimizing dirty data in memory, and instead persisting the writes into the Optane. While large writes would put a substantial stress on the Optane, small writes can be easily persisted there (and larger writes

possibly persisted to SSDs). Thus, in our investigation, we explore three way tradeoff between performance, endurance and data loss.

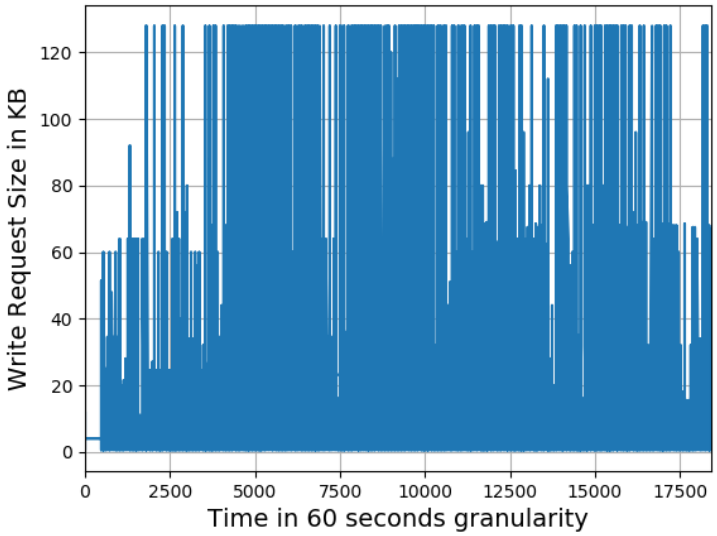


FIGURE 3.1: Write Data Size Variations

The amount of write activity in a workload is crucial from the perspective of both the risk of data loss and endurance impact on the storage device. Unfortunately, HPC systems with large page caches often result in IO behavior that is dominated by writes, and these writes can range over a large size. We observed such characteristics even in the Systor 2017 (Lee et al., 2017) workload trace which is one of the few rather long-duration public traces available. Fig. 3.1 shows the write size distribution for this trace. Note that the overall traffic has substantial percentage of reads, however, the large writes make it difficult to accommodate them in a relatively small ENVM. In particular, simply persisting all writes in ENVM will cause it to thrash. On the other hand, frequent update to write data placed at SSD can wear it out quickly. Thus, performance, endurance and data loss are conflicting goals and thereby require a smart mechanism to alleviate data loss risk by persisting write data immediately between ENVM and SSD such that overall endurance of the system is maintained.

Achieving these tradeoffs in HPC storage, several challenges are involved. First,

ENVM is required to accommodate both read and write data, irrespective of the underlying caching mechanism, adaptive to the workload characteristics to provide substantial performance. Second, the modified data present in ENVM needs to be updated to the backend SSD in an intelligent fashion such that ENVM always has sufficient space to accommodate incoming DRAM evicted read and write data. Third, to maintain the endurance rating it is required to control the writes on endurance sensitive devices intelligently considering their respective endurance characteristics.

The key contribution of our work is the quantification of the benefits of different policies for exploiting ENVM technologies to enhance the performance and endurance of SSD based high performance storage (HPS). The results can be used to design cost-effective and high-performance SSD storage system by pairing low cost multi-bit per cell SSD technologies with relatively small amounts of ENVM. For the workloads examined, we show that our mechanism can achieve 58% read performance and 28% write latency that of native QLC SSD latency by incorporating ENVM that is only $\sim$ 3% that of SSD capacity. However, if the objective is to achieve a full 5-year life-time for overall storage system, the required ENVM capacity can be up to 12% that of SSD capacity.

In addition to the real implementation using Optane and QLC SSD, we also built a comprehensive simulation model for this three-level storage hierarchy. The purpose of the simulation is to examine many corner cases and to also explore parameters that would be very difficult or impossible in a real implementation. In particular, we consider SSDs using 1 to 5 bits/cell. The 5 bits/cell technology (called PLC, or penta-level cell) is still in the works and not currently available. Through our simulator, we also explore how various flash technologies affect the amount of ENVM required to meet the intended objective.

Although various solutions to improve overall performance of HPS are proposed introducing SSD and non volatile memory (Shi et al., 2017, 2019; Torabzadehkashi

et al., 2019; Subedi et al., 2018; Koo et al., 2017; Chen et al., 2011), to the best of our knowledge, a comprehensive study of tradeoff between performance, endurance, and data loss by introducing ENVM in HPS, irrespective of underlying caching mechanism, to keep selective read and write data in the storage hierarchy has not been examined in the literature. With ENVM finally becoming available commercially and the Flash technology moving to QLC with rather poor endurance, it is crucial that we consider combinations of the two to not only address the overall performance and endurance simultaneously, but also exploit the opportunity to minimize or eliminate data loss.

3.2 Motivation and Background

The current trend of ENVM, such as Optane based on PCM technology, STTRAM and etc., are available small in size. The large working set cannot be accommodated totally and thus ENVM is proposed to use as cache to enhance the performance. However, what proportion ENVM needs to be used by read and write data to enhance the performance and maintain the endurance of the overall storage hierarchy while reducing the data loss is never studied before. The write characteristics of the Systor 2017 traces on a busy day (22nd February) during 5 hours busy period for Lun0 as in Fig. 3.2 shows that placing small writes on SSD might wear out the drive fast because of low P/E cycle numbers.

Furthermore, it is necessary to control the number of writes to any endurance sensitive devices. A constant number of allowable writes during any period on these devices might not be of interest. During the high write activity, actual number of writes may exceed the predefined allowable number of writes. Whereas during the non-busy period, the actual number of writes might be lower than the predefined allowable number of writes. Therefore, it is required that the allowable writes should be adaptive with workload write characteristics and device's endurance rating. From

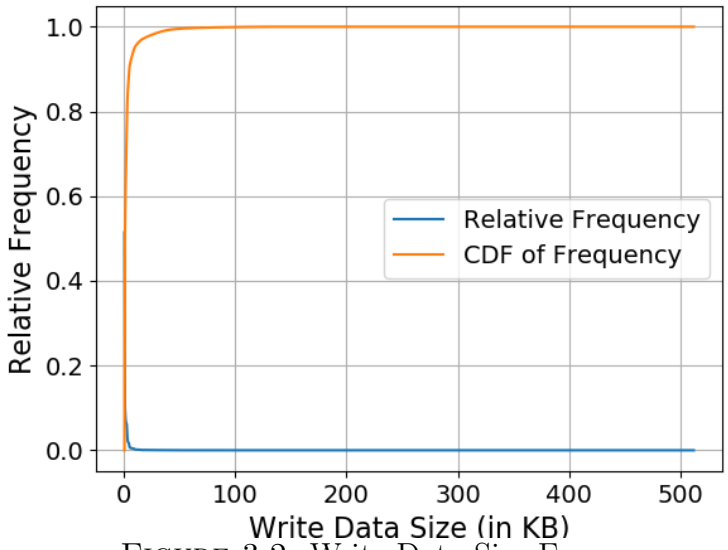


FIGURE 3.2: Write Data Size Frequency

Fig. 3.3, it is observed that there is a homogeneity in the write characteristics of the workload and thus can help in measuring the allowable writes at any time granularity.

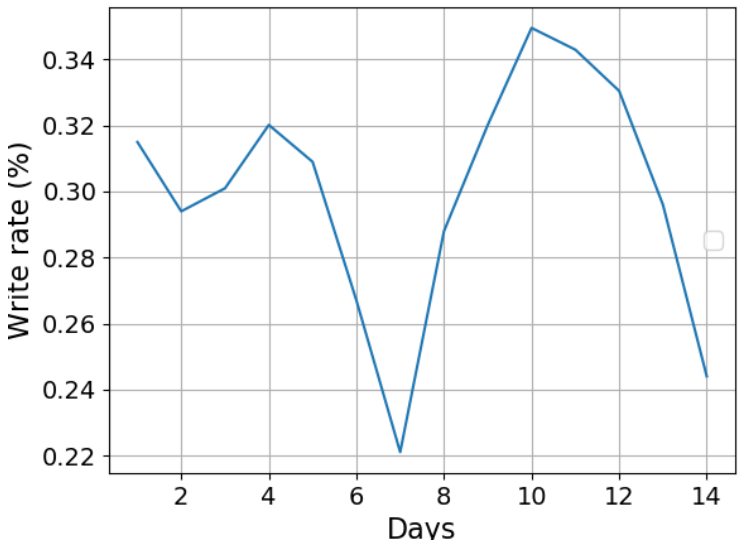


FIGURE 3.3: Write Rate Trend Over a Small Duration

3.3 Current ENVM Enhanced Storage Architectures

In this section, we provide a short overview of the literature on a wide range of solutions to enhance the performance of the system beyond the capability of

conventional non-volatile memory technologies. We also provide a short overview of the literature on life-time optimization of flash-based storage. The performance enhancement of HPS using SSD and ENVM are also studied here and showed distinct difference from our work.

In recent days, SSD and ENVM are studied actively (Niu, 2018; Salkhordeh et al., 2019; Qureshi et al., 2012; Wang and Wong, 2012; Lo et al., 2014; Tai et al., 2018; Gunnam, 2017). Niu et al. (Niu et al., 2018) provide an extensive literature review on the state-of-the-art research for hybrid storage systems relative to algorithms and policies for caching, data migration, and hot data identification.

Ziqi et al. have designed a hybrid cooperative cache, Hibachi (Fan et al., 2017), exploring a DRAM cache as read cache only for clean pages, and non-volatile memory as a write buffer for dirty pages. In our work, we allow the ENVM to automatically store clean pages if the capacity allows doing so.

Lin et al. (Lin et al., 2018) have developed a buffer-aware data migration scheme by exploiting the content information in the buffer cache. They model the process of data migration between HDD and SSD and assess the effectiveness of the buffer-aware data migration scheme to maximize the I/O performance and avoid unnecessary page migrations. In our proposed work, modified data is persisted immediately and writeback is done opportunistically to maximize the I/O performance having in mind endurance requirements for both ENVM and SSD. Tarihi et al. (Tarihi et al., 2016) have developed a design that uses a hybrid DRAM-PCM SSD cache design with an intelligent data movement scheme. This design exploits PCM as a DRAM alternative while alleviating its issues such as long write latency, high write energy, and finite endurance based on thorough I/O characterization of desktop and enterprise applications. Our goal instead is to consider a three level hierarchy in which SSD is used as primary storage.

Lu et al (Lu et al., 2017) have introduced managing performance and endurance

of persistent memory together by reducing the overhead caused due to the strict ordering of writes in persistent memory. They have speculated about inter-transaction ordering and intra-transaction ordering of writes and maintaining consistency with a material alteration to the memory log organization and hardware support at cache level. Garcia et al (García et al., 2017) describe a technique to enhance the life-time of non-volatile memory by reducing traffic towards non-volatile memory and write hot-spots. Write traffic is reduced by different cache replacement policies and with a proposed data compression technique to increase cache capacity. Write hot-spots are reduced by distributing writes over non-volatile memory. Our goal is to address a storage hierarchy and explore the effects of persisting every write immediately based on device characteristics to provide high reliability.

Koo et al (Koo et al., 2017) proposed an optimized progressive file layout method that leverages the advantage of SSDs in HPC cloud environment. In our work we did not touch the SSD file system. Instead we have showed the intelligent data movement can balance the performance and endurance of high-performance storage.

Shi et al (Shi et al., 2017, 2019) have proposed SSDUP and SSDUP+ to use SSD as burst buffer to the underlying HDD as dominant storage device. They have developed a traffic-detection method to detect sequential and randomness in the write traffic to buffer the random writes on SSD. According to SSD's characteristic, random writes tends to wear out fast. In our work, we consider SSD as dominant storage and also maintain its endurance introducing ENVM in HPS hierarchy.

Subedi et al (Subedi et al., 2018) have introduces Intel Optane device in HPC and compared the performance between Optane SSD and NVMe SSD. In our work, we have used Optane SSD as ENVM top of SSD as backend. Intelligent management of data placement at ENVM and SSD helps to enhance the performance and maintain the endurance of HPS hierarchy.

3.4 Enhancing High-performance Storage Using ENVM Technology

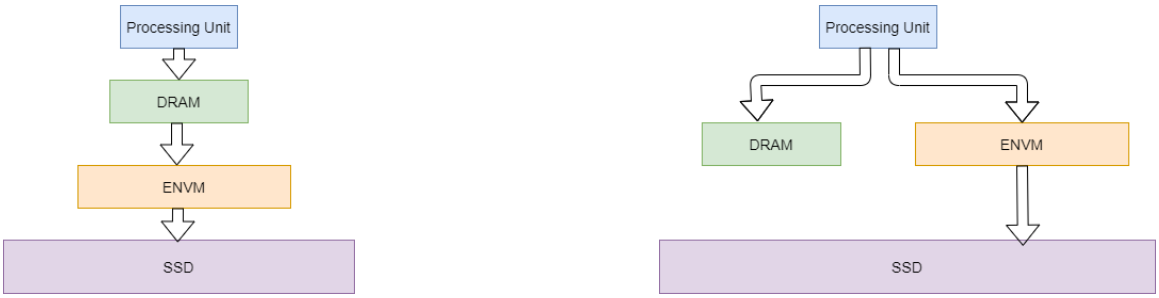
We examine a high-performance storage hierarchy consists of DRAM at the upper level, and ENVM at the middle level atop flash based SSDs in the backend. As in Fig. 3.4, an application can use two logical architectures when the ENVM present in the high-performance storage hierarchy. As ENVM comes with low latency and higher endurance than SSD, it can be used along with DRAM serving different purposes such as another level of cache or as write cache minimizing the data loss. Irrespective of the underlying caching policy, different mechanisms on ENVM can be considered to enhance performance and endurance of the overall system.

3.4.1 Performance Centric Mechanism

A performance centric policy should, at a minimum, minimize over average read and write latency. An additional desirable goal is to reduce tail latency as well; however, this goal could conflict with the goal of reducing average latency.

The DRAM is considered as a simple LRU cache and thus contains the most frequently used data. ENVM is capable of caching all or selective read and write-data evicted from DRAM. Note that it is possible to consider other, more sophisticated, caching policies such as ARC or SARC (Megiddo and Modha, 2003; Gill and Modha, 2005); however, that's not the objective of this line of research and thus we stick to simple LRU based caching.

The write data can be of two types small and large and can be categorized based on the request size. The request size smaller than average write size is considered as small write. Whereas, the request size of average write size and more are considered as large write. The write can happen on ENVM or SSD based on the type of write, small or large. Writeback takes place from ENVM to SSD. However, to make this process efficient, we do not do one element at a time writeback. Instead, we invoke



(a) Next Level Cache (b) Write Cache

FIGURE 3.4: High-performance Storage Logical Architectures

the writeback opportunistically based on the IO queue depth of the SSD. Obviously, an evicted block must stay in ENVM until the writeback is done; therefore, there is a tradeoff between opportunism and making space in the ENVM. The opportunistic writeback is achieved by predicting the gap and scheduling writeback I/O accordingly.

Both lists are maintained at ENVM level and potential move of elements between them only changes the pointers (or metadata, in general).

Keeping track of which data is persisted in ENVM is necessary. Therefore, we assume that there is a journaling file system that treats each data element as a file and thus is responsible for maintaining the mapping necessary to know its' physical location on backend storage. Our priorities are to obtain high-performance and retain desired endurance. Therefore an amount of ENVM should still be reserved for this journaling. How much extra space is needed to keep this metadata depends on the size of data elements, but it is nevertheless a fraction of the size of the area that is devoted to persisting.

3.4.2 Endurance Centric Mechanism

It is necessary to control the number of writes to the endurance-sensitive device to respect the endurance rating of it. The ENVM can be used to accommodate all or small writes as required and others are accumulated in SSD. Endurance centric mechanism allows write-persist to the device based on an estimated write-budget

under the experienced traffic characteristics at a suitable time granularity. The write budget (W_{est}) is calculated based on corresponding device's endurance rating. The traffic characteristic (TR_{est}) is aggregated from the past using exponential smoothing as in Equation 3.1.

$$TR_{est(t+1)} = \alpha * TR_{act(t)} + (1 - \alpha) * TR_{est(t)} \tag{3.1}$$

Algorithm 1 Write-Budget Estimation

WriteBudget (reward R)
$W^{Week}_{est(t+1)} \leftarrow W^{Week}_{est(t)} - R$
for $Day \leftarrow Monday \rightarrow Sunday$ **do**
    $TR^{Day}_{est(t+1)} \leftarrow \alpha * TR^{Day}_{act(t)} + (1 - \alpha) * TR^{Day}_{est(t)}$
$TR^{Week}_{est(t+1)} \leftarrow \sum_{Day} TR^{Day}_{est(t+1)}$
for $Day \leftarrow Monday \rightarrow Sunday$ **do**
    $W^{Day}_{est(t+1)} \leftarrow W^{Week}_{est(t+1)} * \frac{TR^{Day}_{est(t+1)}}{TR^{Week}_{est(t+1)}}$

The write-budget estimation procedure should follow the workload characteristics and adapt to the change in workload over time. The estimated write-budget should be adaptive; high enough during the write-busy period to accommodate maximum possible large write-persist at the SSD level and thus provide ENVM the opportunity to control the space for immediate persisting. The Write-Budget estimation procedure is represented in Algorithm 1. Given the capacity and endurance of the SSD, a coarse-grain write-budget is calculated initially. According to the workload characteristic the write budget is distributed over finer time granularity and the surplus amounts of writes at coarser granularity are adjusted over the next cycle. We consider a week as the time period with coarser time granularity and a day as the time period when we consider finer time granularity. The adjustment of excess or surplus is calculated by measuring the difference between actual number of write-data persists at the device over a week (W^{Week}_{act}) and the estimated number of write-data persists in the other device over a week (W^{Week}_{est}) as Equation 3.2. The surplus is adjusted by measuring

the actual number of writes on the device over a week (W_{act}^{Week}) and the estimated number of writes on the device over a week (W_{est}^{Week}) as Equation 3.2. This endurance centric mechanism can be applied considering any of the devices. We applied this mechanism on SSD and Optane to enhance the SSD life-time.

$$R = \sum_{Day} W_{act(t)}^{\text{Day}} - W_{est(\text{t})}^{\text{Week}} \tag{3.2}$$

3.5 Proposed Policies

We consider three different policies in managing the high-performance storage hierarchy presented in Fig. 3.4 including ENVM serving different purposes such as another level of cache or as write cache minimizing the data loss. The performance and endurance centric mechanisms are applied to these policies accordingly. The first policy is "Oblivious Caching" without regard to clean or dirty data. The second and third policies make this distinction, but the second policy does not do immediate persisting of all modified data in ENVM, whereas the third policy does. These policies are described in more detail in the following.

3.5.1 Policy-I: Oblivious Cache (OC)

In this policy, the ENVM is used as another level of independent larger cache next to DRAM cache. As ENVM has lower latency than SSD, the average read and write latency can be improved. However, since this policy does not limit the amount of modified data in DRAM or how long it resides there before being evicted and written back to ENVM or SSD, the policy carries data loss risk due to system crashes. We assume that both the ENVM and SSD implement suitable mechanisms to maintain consistency in the event of system crash. Since the ENVM is used as a DRAM-like cache, it can use a simpler mechanism to ensure all or nothing semantics for the data and data structures to manage the cache. Assuming that the SSD implements a file

system, appropriate journaling mechanism would handle consistency issues for it.

The performance of this policy will vary with different caching policies, but our focus is not caching policies per se. Therefore, we will use simple LRU or a LRU variant for both DRAM and ENVM level. All data evicted from the DRAM caches goes into the ENVM cache but evictions from the ENVM can be discarded if they are clean. Evictions of modified blocks from ENVM must be written back to SSD. Note that this policy does not explicitly consider the endurance of SSD or ENVM while doing the writes.

In this policy, a lookup operation travels first to DRAM, next on a cache miss to ENVM and finally resolves at SSD upon missing out of the ENVM. On DRAM eviction all data are stored in ENVM cache. If the number of modified data touches the concurrency threshold, prioritized writeback writes data to the SSD. Otherwise, an opportunistic writeback is done.

3.5.2 Policy-II: Next Level Cache Without Immediate Persisting (NLCWOIP)

In this policy also the ENVM is used as the next level cache on top of the SSD as backend. The data evicted from the DRAM cache to be written to the ENVM prioritizes write data over read data unlike Policy I. A portion of ENVM, adaptive to the workload characteristics, is reserved for storing relevant clean data for future use for lowering the read latency. Like oblivious cache, this policy also risks data loss due to system crashes due to same reason.The consistency and system crash handling is also same as Policy I.

The performance of this policy is enhanced by incorporating opportunistic and conditional writeback mechanism like policy I. Also, speculating relevant clean data for storing in ENVM avoids unnecessary evictions from ENVM Cache (and hence writebacks to the SSD). The data in DRAM with higher the access frequency acquires higher chance to be stored in ENVM. The selection of relevant data involves very

simple computation and hence does not add much to the latency.

In this policy, DRAM Cache miss works similarly as Policy I. Upon eviction, all modified data are stored in ENVM, whereas only the clean data selected as relevant gets stored. Also conditional and opportunistic writeback mechanisms are involved.

3.5.3 Policy-III: Next Level Cache With Immediate Persisting (NLCWIP)

In this policy, we persist all modified data immediately either to ENVM or SSD. In this policy the DRAM works as read cache for clean data whereas the ENVM works primarily as write cache for the SSD. The ENVM is also used as extended read cache adaptive to the workload characteristics. The non volatile and low latency characteristics of ENVM allow it to work as a write cache alongside the DRAM. Moreover, adapting to the workload characteristic, the low latency aspect of ENVM makes it suitable for storing relevant clean data evicted from the DRAM cache. This policy omits the data loss risk by persisting modified data immediately in the ENVM cache or SSD.

The performance of this policy is enhanced by incorporating conditional writeback along with opportunistic writeback mechanism to have available space for modified data to persist immediately. Also speculating relevant clean data for storing avoids unnecessary eviction from ENVM Cache. Both of these techniques help to lower the read and write latency. Unlike the other two policies, the policy III incorporates endurance centric mechanism. There is a common phenomenon that small writes appear much more frequently than large writes. The large and small both write affects device life-time based on their endurance characteristics. Thus to maintain the endurance of the ENVM and SSD, the small writes are persisted in ENVM and large writes in SSD. Also intelligent write-budget procedure is applied to maintain the endurance rating of the considered device.

In this policy, a read miss results in ENVM lookup and on success data are accessed

from there or from SSD otherwise. In case of write hits in ENVM, writes are persisted there. For write miss, if the write data is large, it is persisted on SSD respecting the write budget. For small writes, it is persisted in ENVM always, DRAM eviction and writeback mechanisms are same as described earlier for other policies.

3.6 Experimental Evaluation

3.6.1 Implementation Setup

In order to explore the proposed mechanisms, we built an experimental setup consisting of an available Optane SSD module and a SSD. Both were of rather modest size; however, they are adequate to explore the issues of highly variable and write intensive workloads that HPC systems migh face. At the time of our experimental setup, the available Optane modules were only 32GB, and we paired one of those with an Intel 1 TB QLC SSD and 1 TB Samsung EVO TLS SSD. The DRAM buffer was sized accordingly. This was done on an Intel 6-core i5-8500 CPU @ 3.00GHz machine. Because of small size of Optane we limited DRAM cache to 4GB. The implementation intercepts the IOs and forces them through our own DRAM cache (different from system's buffer cache). The Optane SSD does not necessarily require a file system and we used it without one, exactly like memory. The metadata relating to the Optane was kept mostly in DRAM for efficiency; however, there is adequate information in the blocks stored in Optane so that its contents can be reconstructed in case of a machine crash. The SSD hosts a normal EXT4 Linux file system with normal journaling mechanisms. We drove this system by generating IO requested based on several storage IO traces.

The endurance rating measured in GB per day are 100, 110 and 252.5 GB for Optane, QLC and TLC respectively.

The crash management of the system is considered as follows. For DRAM, it is expected that the normal data limiting mechanisms will be in place; however, for

the purposes of these experiments, no such restrictions played. Ensuring the all or nothing semantics for a single write is much simpler in our case, particularly due to the availability of Optane, where the metadata are recorded for only the ongoing write to recover from its interruption. With Optane used as a memory cache without a FS, it involves a rather trivial metadata recording. For policy III, the journaling system on the SSD file system comes into play only for evictions from Optane. It is even possible to simplify journaling for SSD FS (since the data is safely stored in Optane and can persist there until the eviction is properly handled in its entirety). However, SSD FS is not touched in this work.

Additionally, a comprehensive simulation model for three-level storage hierarchy built. The key purpose of this simulator was to consider PLC and how it affect the amount of ENVM required to meet the target.

3.6.2 ENVM Specific Implementation Detail

To accommodate both read and write data ENVM maintains two separate lists clean list (CL) and modified list (ML). Both lists are maintained at ENVM level and potential move of elements between them only changes the pointers (or metadata, in general).

Two seperate I/O queues: demand queue that serves the read request and writeback queue are considered. In general, the demand queue gets higher priority, but empty demand queue triggers opportunistic writeback if there is enough time estimated through gap prediction Gap_{est} (Equation 3.3) - the exponentially smoothed value of the amount of time between two requests with $\alpha = 0.5$ so that estimated and actual have same influence. The number of data elements that are selected to be written back $\text{WB}_{\text{element}}$ is calculated from average access latency and the estimated Gap_{est} as shown in Equation 3.4.

$$\text{Gap}_{\text{est(t+1)}} = \alpha * \text{Gap}_{\text{act(t)}} + (1 - \alpha) * \text{Gap}_{\text{est(t)}} \tag{3.3}$$

$$WB_{\text{element}} = \frac{\text{Gap}_{\text{est}}}{\text{SSD average access latency}} \tag{3.4}$$

Also, conditional prioritized writeback functionality is used to ensure that there is enough space for the ML. The concurrency threshold is defined in equation 3.5. The conditional writeback is triggered when the ML size exceeds hysteresis percentage applied on the concurrency (S).

$$\text{S} = \frac{\text{Write miss count}}{\text{Write miss count + Evict miss count}} \tag{3.5}$$

To determine the relevant data evicted from DRAM and to store in the CL follows a simple frequency based mechanism in this context. A clean evicted data element will be selected as relevant data if they have more access frequency during DRAM life-time than any element currently lying on ENVM CL.

3.6.3 Write Budget Specific Implementation Detail

Based on traffic characteristics, the write budget is estimated using Algorithm1. The smoothed value with α being 0.5 is used as in Equation 3.1 to consider same influence for estimated and actual value of traffic experienced.

3.6.4 Evaluation Metrics

With performance-centric measures, we would like to gauge how effective our policies are to attain desired performance. Whereas in case of endurance, we have measured effectiveness of the write traffic control at the endurance-sensitive device. Therefore, different metrics are considered for measuring data loss, performance and endurance as described below.

Data loss metric:

- Average Data loss Risk: Average amount of modified data lying in DRAM, which can be lost due to system failure, is measured in .

Performance centric metrics:

- Average Read Latency ($Avg_{\mathbf{RL}}$): Weighted average of DRAM read latency (L_D), ENVM average read latency (L_{NR}) and SSD average read latency (L_S) together with latency (L) due to DRAM eviction, writeback to SSD from Optane defined as Equation 3.6 and measured in microseconds.

$$\begin{aligned} Avg_{\mathbf{RL}} &= H_D * L_D + H_N * L_{NR} \\ &+(1 - H_D - H_N) * L_S + L, \end{aligned} \tag{3.6}$$

  where H_D and H_N are the number of time DRAM and ENVM are accessed.

- Average Write Latency ($Avg_{\mathbf{WL}}$): Weighted average of DRAM access latency (L_D), ENVM write latency (L_{NW}) and SSD average access latency (L_S) together with latency (L) due to DRAM eviction, writeback to SSD from Optane defined as Equation 3.7 and measured in microseconds.

$$\begin{aligned} Avg_{\mathbf{WL}} &= H_D * L_D + H_N * L_{NW} \\ &+(1 - H_D - H_N) * L_S + L, \end{aligned} \tag{3.7}$$

  where H_D and H_N are the number of time DRAM and ENVM are accessed.

- Tail Read Latency: 99% latency experienced during read operation for various sizes of requested data and measured in microseconds.

Endurance centric metrics:

- ENVM Write: Amount of data writes on ENVM and measured in GB.

- SSD Write: Amount of data writes on SSD and measured in GB.

3.6.5 Dataset Characteristics

In order to study endurance and data loss issues relevant to HPC, we sought storage system traces that are relatively write heavy and have significant variation in write sizes. Unfortunately, most of the publicly available traces or the benchmarks such as SpecSFS do not provide trace with these characteristics. For example, we considered the five different SRCMap storage workloads in (Verma et al., 2010) and available at SNIA's website. However, due to absence of large variation in request size, they are not interesting for testing and thus not reported results for this workload here. The Systor 2017 trace (Lee et al., 2017) is a much longer trace and consist of wide variations of request size. This workload represents virtual desktop infrastructure (VDI) where a number of users log in remotely and use typical desktop applications (e.g., Microsoft Word, Excel, etc.). A separate VM is created for each user and is given resources depending on the user's requirements. The workload is generally read intensive but has huge variations of write request size from 512 bytes up to 128KB. This trace considers 2706 I/O traces from 6 different block storage devices (LUNs

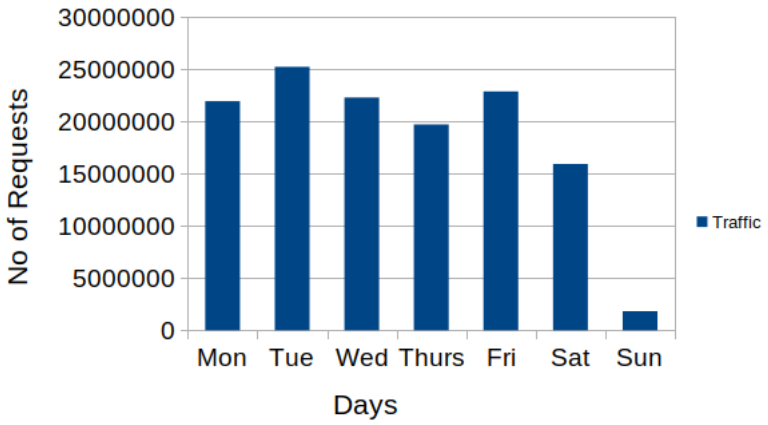


FIGURE 3.5: VDI Average Traffic Characteristics

0,1,2,3,4, and 6), each of them experiencing almost the same workload behavior. Anyone of them can be representative of the set of LUNs and we have considered LUN0 in this context. The typical week long experienced traffic characteristics of

this trace are shown in Fig. 3.5. Based on the number of requests experienced per day, a day long traffic can be classified as busy, moderate and low traffic day.

3.6.6 Results And Discussion

We have evaluated our proposed policies by measuring the evaluation metrics for performance, endurance and data loss. Also, we have compared our proposed policies with a base policy configuration comprising of DRAM as the cache to the SSD without any ENVM.

3.6.7 Data Loss Evaluation

The average data loss risk is measured in KB during busy, moderate and low traffic days for all policies. It is clear from Fig. 3.6 that both policies I and II experience equal average data loss risk. In contrast, Policy III has zero average data loss risk due to immediate data persist.

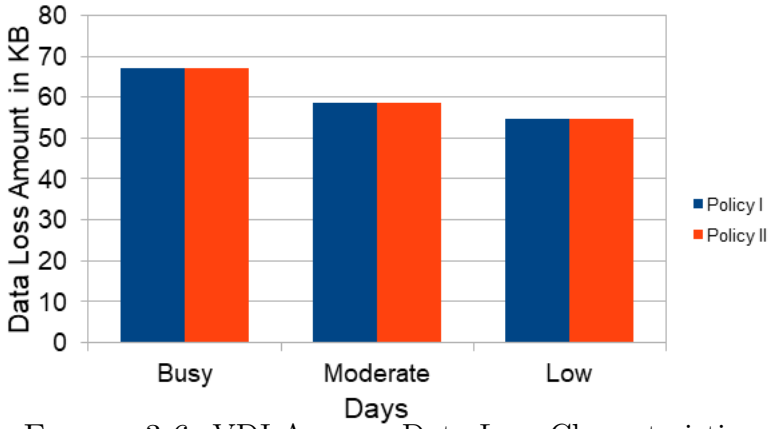


FIGURE 3.6: VDI Average Data Loss Characteristics

3.6.8 Performance Evaluation

The figures 3.7 and 3.8 represent average read and write latency for three different policies during busy, moderate and low traffic conditions. The comparative study uses 1 TB QLC and Optane of size 32Gb, which is $\sim$ 3% of the size of the QLC.

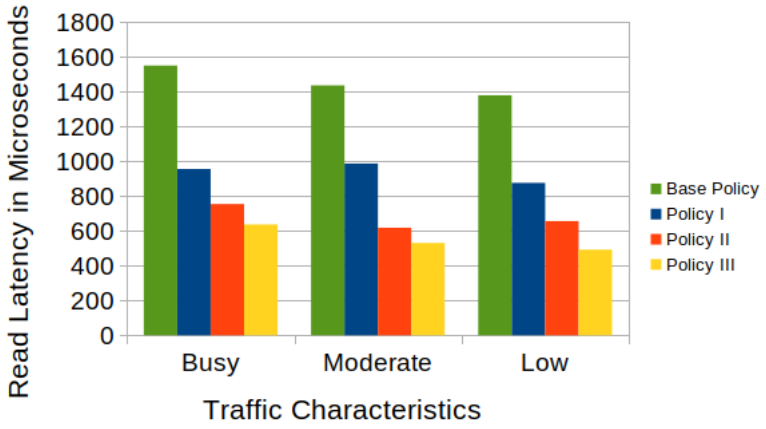


FIGURE 3.7: VDI Average Read Latency

The Fig. 3.7 shows that the absence of ENVM results in much higher read latency than any of our proposed policies. The frequent access to SSD on DRAM miss is the reason such high latency is observed. Policy I improves the latency by caching the DRAM evicted elements, whereas Policy II improves it even more by caching the relevant DRAM evicted elements that lowers the number of unnecessary eviction from ENVM. Among our proposed policies, the Policy III works the best, improving latency by 58% as compared to the base policy for busy traffic, as it leverages the whole DRAM and a significant portion of ENVM to cache the read data. Policy III also eliminates data loss by immediately persisting all writes. In contrast, even though Policy II gets to use more ENVM for read data (by virtue of not having to persist every write), it falls short because of modified but infrequently used data lying in DRAM. This aspect obviously depends on the workload characteristics and speed of DRAM relative to that of Optane.

The Fig. 3.8 shows that the absence of ENVM brings huge write latency as the modified data upon eviction from DRAM is written directly to SSD. For Policy I and II, all writes are done at DRAM and hence results in much lower write latency than Policy III, where small writes are persisted on ENVM and large writes on SSD.

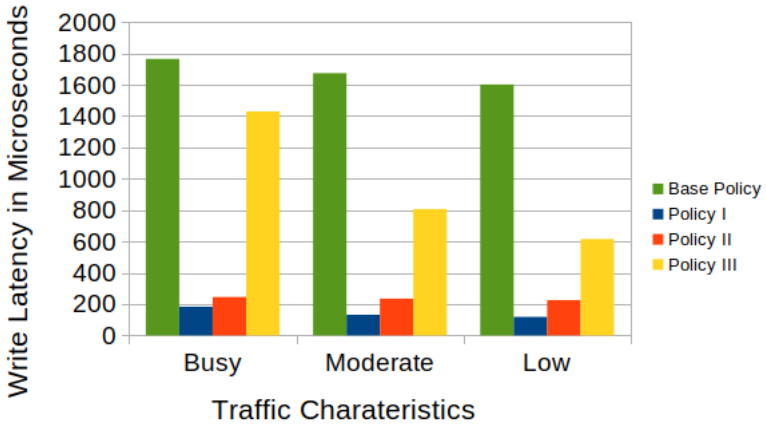


FIGURE 3.8: VDI Average Write Latency

During the busy period, large writes appear more than moderate and low traffic days. Therefore, Policy III experiences more write latency during busy traffic than moderate and low. However, Policy III improves 28% latency during busy day than base policy. Policy II experiences more latency than I because of a greater number of writebacks from ENVM to SSD.

As tail read latency for different request sizes can provide more insight, we have measured 99% read latency experienced for three different policies during busy traffic. The request sizes in the workload are starts from 512B and go up to 128KB. Here we have considered request size multiple of 4KB. The 99% read latency for various request size for three different policies are shown in figures 3.9. All three policies yield approximately the same read tail latency, which argues for policy III since it eliminates data loss. The tail latency without ENVM (not shown) will obviously correspond to going directly to SSD and will be very high. The tail latency obviously increases with the transfer size in all cases.

3.6.9 Endurance Evaluation

Under the base policy, the VDI workload can degrade the endurance rating of the SSD due to large write bursts. For this, we measure the number of writes that exceed the daily write limit for the SSD. We call these as **critical writes** and measured in

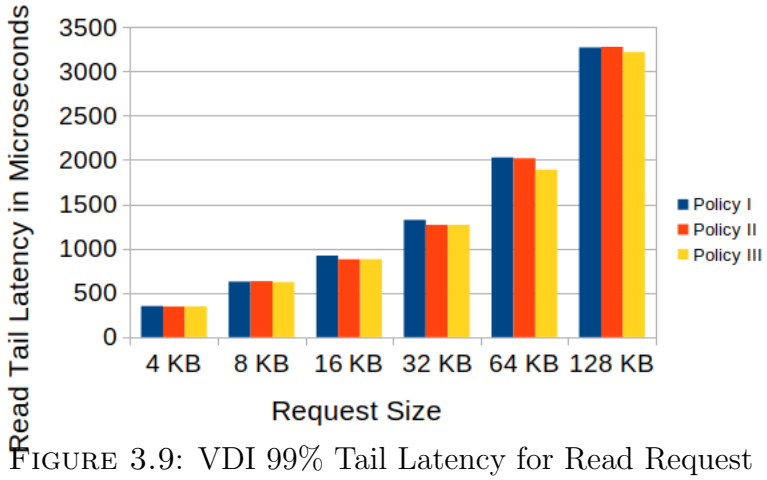


FIGURE 3.9: VDI 99% Tail Latency for Read Request

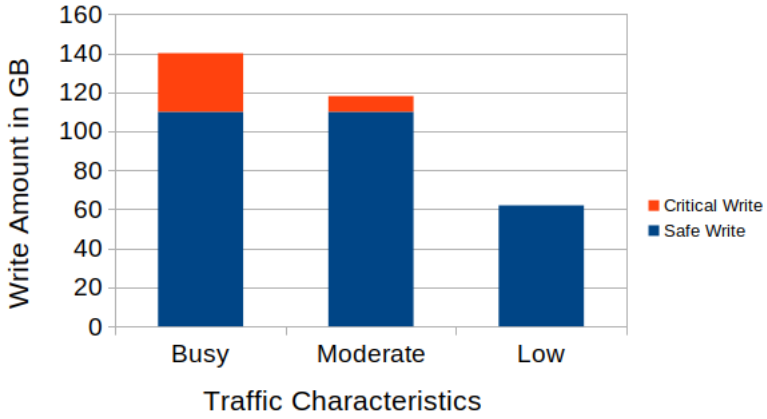


FIGURE 3.10: VDI SSD write for Base Policy

GB. Writes below this limit are called **safe writes** and measured in GB. Fig. 3.10 shows the safe writes and critical writes for the base policy. This figure depicts that a significant number of critical writes are experienced during busy and moderate traffic. It is desired to limit the amount of write data on SSD.

Fig. 3.11 shows the observed SSD writes for all the three policies. In Policy I and II, all writes are initially done at DRAM and frequent writes on same DRAM elements reduces SSD writes though risk of data loss remains. For both of these policies, the ENVM accumulates all or selective data evicted from DRAM and eventually passes modified data to the SSD during writeback. In case policy III, write budget scheme is applied on SSD and large writes always go to the SSD. These are in addition to

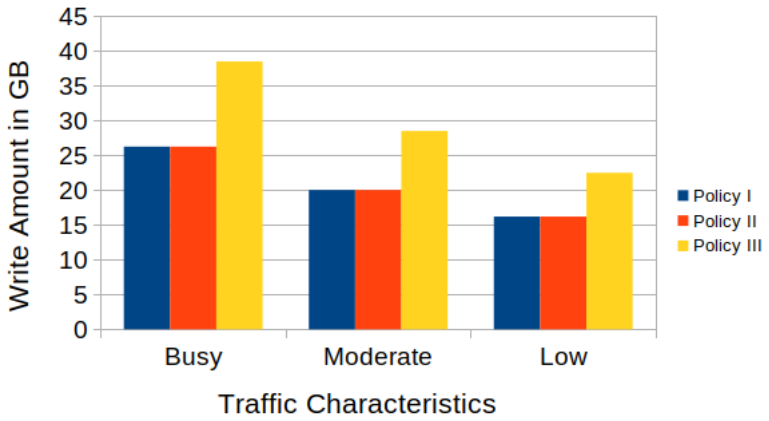


FIGURE 3.11: VDI SSD write amount for different Policies

the writeback from ENVM. Therefore, amount of SSD writes are more than Policy I and II and does not exceed the safe write limit showed in Fig. 3.10.

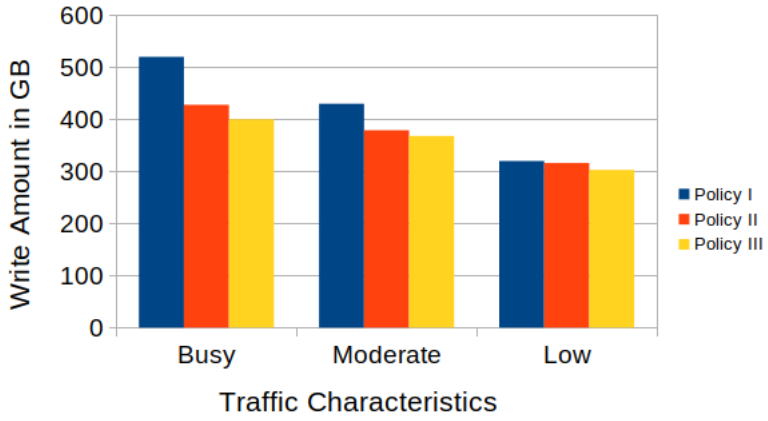


FIGURE 3.12: VDI ENVM write amount for different Policies

Fig. 3.12 shows the total number of ENVM writes for all three policies. In case of policy I and II, ENVM writes are much higher than Policy III. In case of policy I, a larger number of writes on ENVM happen because of caching all DRAM evicted read and write data. Policy II does better but still accumulates both small and large writes in ENVM on eviction from the DRAM. Policy III works the best and lowers writes by 22% than Policy I by persisting only small writes and storing relevant read from eviction from DRAM. In fact, all three policies violates the ENVM endurance rating,

which is 110 GB per day for 32 GB Optane. Thus, to maintain endurance rating for both ENVM and SSD, one needs a careful balancing act which can be achieved not only by Policy variation but also by controlling the size of the ENVM and SSD. In particular, by making the ENVM about 12% of the SSD capacity, we can maintain endurance of both for all policies for this workload.

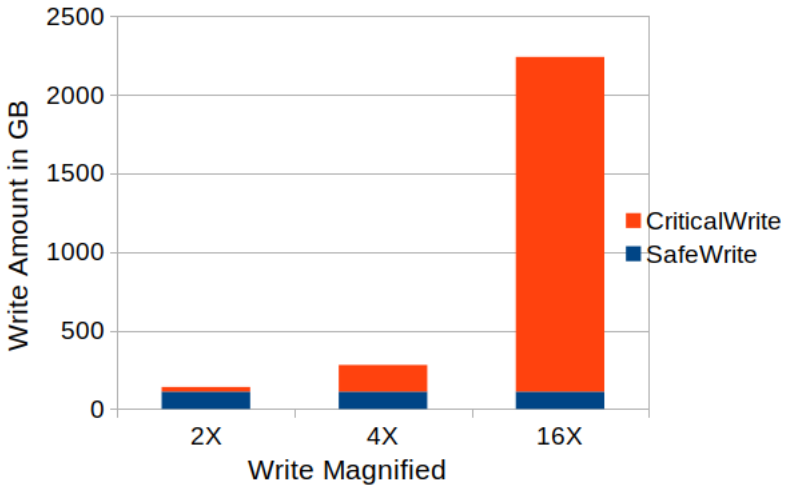


FIGURE 3.13: SSD writes for Base Policy with Magnified Trace

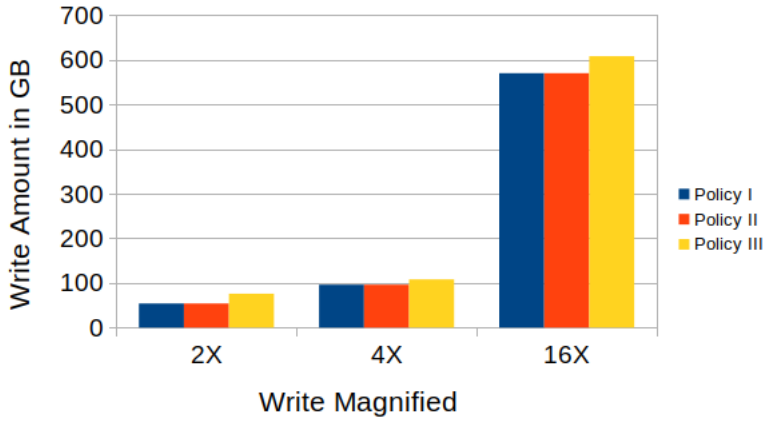


FIGURE 3.14: SSD writes for different policies with Magnified Trace

Given the lack of publicly available HPC storage traces, we modified the Systor Trace to incorporate large write feature, typical of HPC. The write sizes in Systor

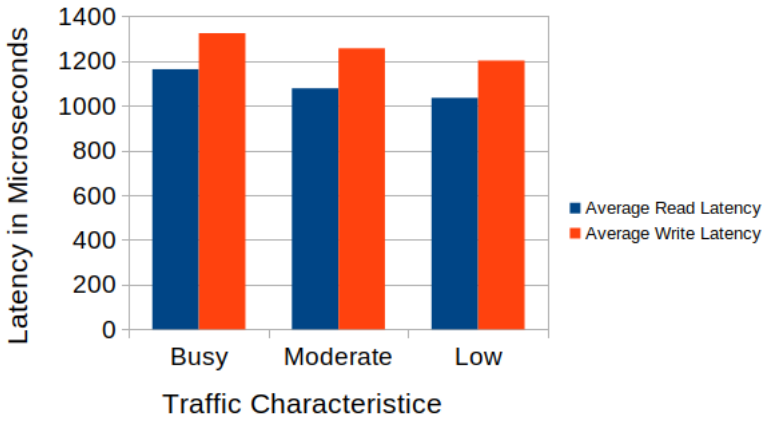


FIGURE 3.15: Average Read & Write Latency for TLC with Systor Trace

trace are magnified to two (2X), four(4X) and sixteen(16X) times. Fig. 3.13 shows the amount of safe and critical writes on SSD due to different large writes during busy period when base policy is applied. It is clear that the SSD endurance rating is affected substantially for 4X and 16X cases under the baseline policy.

Next we examine how three policies impact SSD writes for different magnified write traces. This is shown in Fig. 3.14. For 2X magnified trace, all policies still results in safe write on SSD. However, for 4X modified trace, policy III results in small amount of critical write. In case of 16X magnified trace, all these three policies results in huge amount of critical writes. This suggests to have a bigger amount of SSD with high endurance rating to accommodate large amount of writes.

To examine the behavior of different policies, we also consider Samsung 860 EVO TLC SSD along with DRAM without considering ENVM. The capacity of TLC is restricted to 75% capacity of 1TB QLC assuming the same number of cells. In case of the VDI workload, the endurance is not violated by persisting all writes on TLC SSD directly. The average read and write latency experienced, as in the Fig. 3.15, is much more than the latency of QLC pairing with ENVM as explained in figures 3.7 and 3.8. Thus, inserting Optane in the storage hierarchy is still useful from the perspective of performance while limiting data loss.

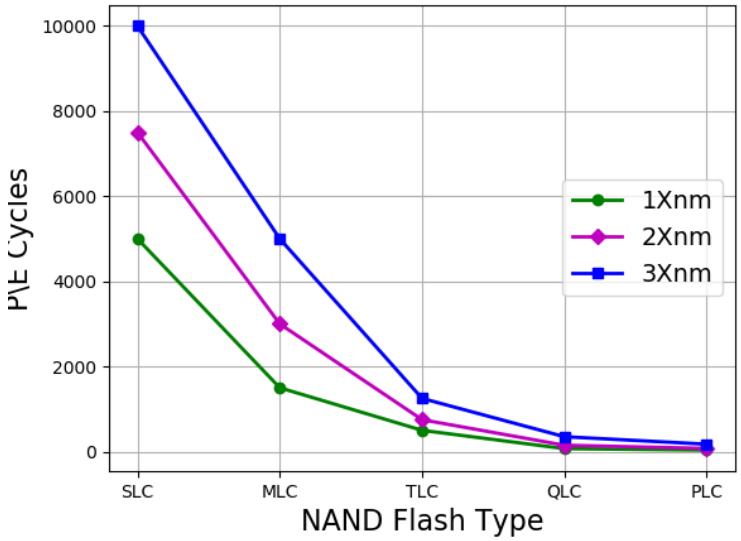


FIGURE 3.16: P/E Cycles vs. bits/cell

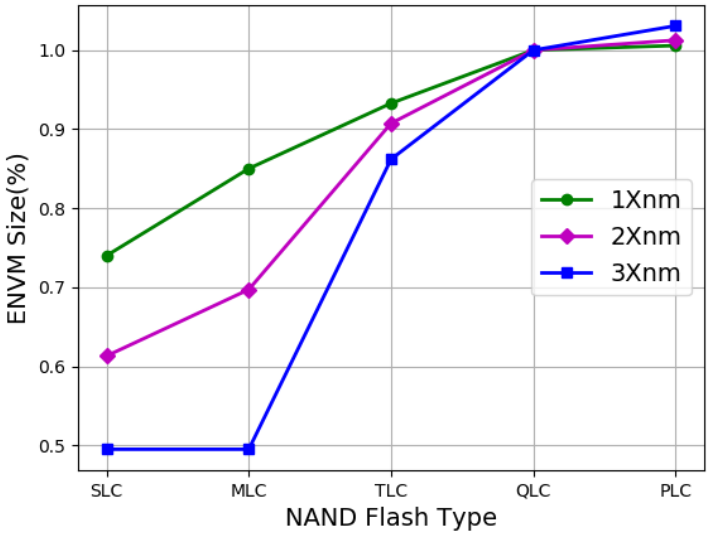


FIGURE 3.17: Estimated ENVM cost to maintain same Endurance

As we approach towards more bits per cell and cell types in NAND Flash technology, 5 bits per cell (PLC), it is interesting to investigate the estimated ENVM need to maintain the endurance rating. We simulated policy III during considered busy write traffic trend for different flash technologies to

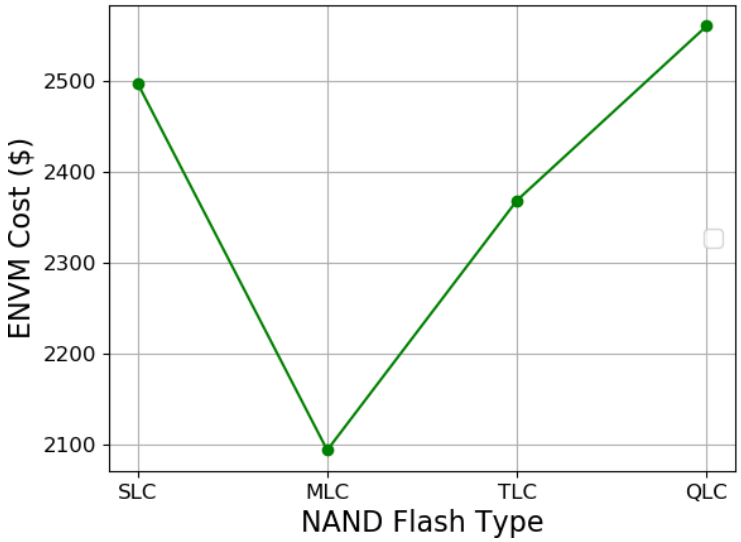


FIGURE 3.18: ENVM vs. Flash Technologies

compare the estimated ENVM sizes for absorbing critical writes corresponding to SLC, MLC, TLC, QLC and PLC based on the values publicly hosted (*https://blocksandfiles.com/2019/08/07/penta-level-cell-flash/*). The capacities of SLC, MLC, TLC and PLC are considered as 25%, 50%, 75% and 125% capacity respectively of 1TB QLC assuming the same number of cells. The P/E cycles for different flash technologies and cell types are considered as publicly hosted. The ENVM required for different flash technologies and cell types are represented in the Fig. 3.17. For different flash technologies, SLC requires smallest amount of ENVM to maintain the overall endurance compared to MLC, TLC, QLC and PLC. With different cell types, 1Xnm requires more ENVM for SLC, TLC, QLC and PLC than 2Xnm and 3Xnm. The Fig. 3.16 shows the p/e cycle for different flash technologies and cell types. Though PLC gets much more blocks to be written than SLC, The sharp drop in P/E cycle from SLC to PLC is not compensated by the relative increase in the number of blocks to be written and thus overall TBW drops consistently with the technology progress from SLC to PLC. In other words, so long as the

SSD is large enough to hold workloads data, having more bits/cell hurts in terms of endurance. However, the required ENVM comes with price. The Fig. 3.18 represents the estimated price associated with required amount of ENVM and same capacity of different SSD flash technologies. It is noticeable that the combination of MLC and ENVM is much lower than the other combinations.

3.7 Conclusions

In this work, we presented the opportunities and challenges in using emerging NVM technologies to manage the tradeoffs between performance and endurance characteristics of low endurance SSD based high-performance storage. The availability of high speed technologies like Intel Optane also allows immediate persist of modified data and thereby eliminate (or at least substantially reduce) data loss potential and also simplifies resilience. We discussed several ways of combining NAND flash technologies and ENVM technologies. The immediate persist actually reduces the read latency, which is often more critical than the write latency. Of course, the overall write latency goes up (since all writes must eventually go to the SSD) but this is not important since a write completion response can be returned to the application as soon as the ENVM persist is done. The high-performance computing applications often involve large writes and the real price of immediate persist is in potentially degrading the endurance of the ENVM. However, since EVNM has very high PE cycle rating, a somewhat larger size can often address the problem.

In the future, we plan to explore other policies for storage hierarchies involving SSD and ENVM, including byte addressable ones, and will also explore performance for a larger set of workloads.

CHAPTER 4

ACHIEVING DETERMINISTIC LATENCY IN SHARED ENVIRONMENT

4.1 Introduction and Motivation

The ongoing replacement of hard drives (HDDs) by solid-state drives (SSDs) with NVMe (Non-Volatile Memory Express) interface in the enterprise has enabled much higher performance and low end-to-end storage latencies (Micheloni et al., 2013; Cobb and Huffman, 2012; Huffman, 2012). However, the tail latency in current SSDs can range up to several milliseconds.

For example, Fig. 4.1 shows the latency distribution of a recent Intel NAND SSD device (Intel DC P4610 SSD) with a workload of 70%-read and 30%-write of 4KB blocks (Int, 2020). It shows that the 99 percentile tail latency can be more than 1 ms (This figure is only for illustration; our experimental setups don't use this SSD and the workload is different too). Popular applications such as streaming applications, social media platforms, financial transactions require not only a low average access latency but also a short tail latency so that transaction to transaction latency does not vary substantially (Chen et al., 2019). Thus, tightly controlling the tail latency is essential for emerging storage systems (Xu et al., 2013; Dean and Barroso, 2013).

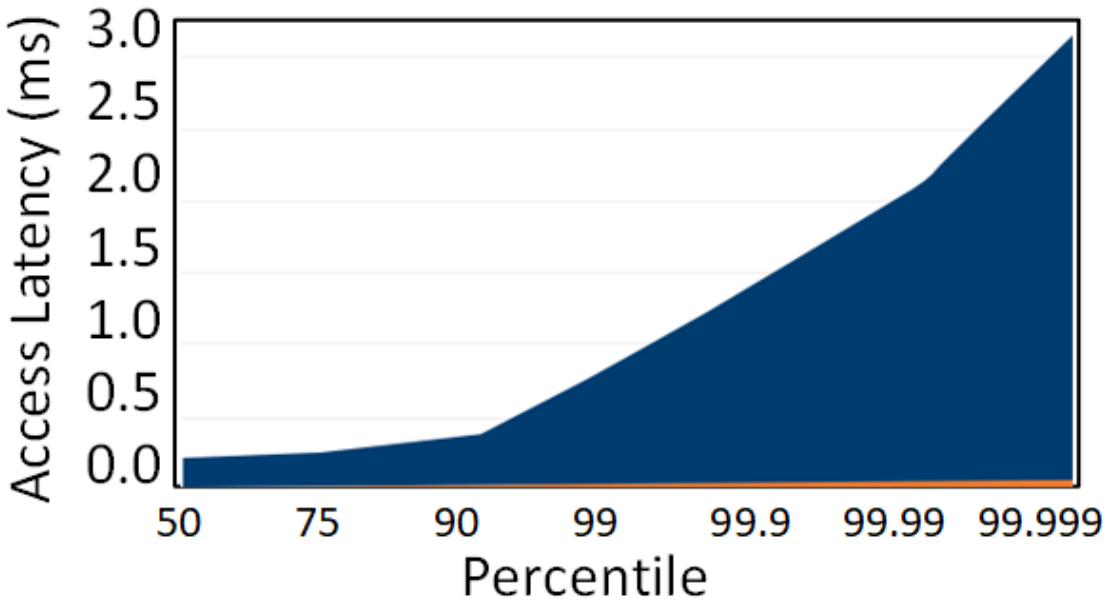


FIGURE 4.1: Latency Distribution for Intel DC P4610 SSD

The NVMe 1.4 specification (NVMe v1.4) introduces the predictable latency mode (PLM) (NVM, 2019) feature for NVMe SSDs to achieve deterministic latency. The PLM feature is based on the concept of *deterministic window* or DTWin and *non-deterministic window* or NDWin modes. During the DTWin mode, the storage access latency is made deterministic by avoiding background activities, which are deferred until the SSD goes into NDWin mode. The DTWin mode is realized by the limited DTWin budget (explained in sec 5.2.2). A careful DTWin budget allocation becomes critical if (a) several applications, with different QoS classes, have tight tail-latency requirements and thus need to take advantage of the PLM, and (b) the data accessed by these applications are located on the same SSD.

If the workloads are running on the same host, the host itself can regulate the share of the DTWin budget for each application; however, when multiple hosts target the same SSD, a coordinator is needed. In this work, we design and evaluate such a coordinator, henceforth called PLM coordinator (PLMC).

A typical data center environment concentrates storage in a small number of

"storage servers" that are accessed by all the hosts; therefore, the scenario of multiple hosts accessing data on the same SSD is routine. It is likely to even increase as SSD sizes move from the current few TB to few tens of TB. Typical examples of such shared access include traditional databases, semi-structured key-value stores, or document stores containing documents, images, videos, etc. If multiple workloads compete for a limited DTWin budget, providing each of them a proper QoS can become quite challenging.

The existing PLM mechanism, as currently defined, focuses only on read IO latency improvement, and hence this paper is also focused on read I/O management. It calls for buffering of all writes that arrive during a DTWin period in an NVRAM device (intended to be internal to the SSD but could also be external) and flush them to the SSD during the NDWin period. Thus while writes are very important to the overall PLM scheme, they are out of scope for this paper and will be addressed in future.

To the best of our knowledge, there is no other comparable mechanism in the literature at this point. Specifically, our contributions are as follows:

- **Simulating PLM feature**: Since the PLM capability is nascent and still not available in any commercial SSDs, we need to lean on simulation to examine PLM performance. For this, we have modified MQSim (Tavakkol et al., 2018b), which is a comprehensive and validated model of NVMe and SATA-based SSD.

- **Extending PLM feature**: The PLM feature is currently defined only for use by a single host. In this work, we extend it to multiple hosts sharing an SSD by defining a *PLM Coordinator* (PLMC) that interacts with the hosts and allocates a suitable DTWin budget to each. The PLMC resembles any other NVMe host and thus does not require any changes to the existing protocols. Therefore, PLMC can be implemented in a container or VM in the storage server itself

that the hosts can invoke.

- **PLMC Evaluation**: The proposed PLMC predicts the traffic of different classes and uses it to allocate the DTWin "counts." We show that this mechanism can achieve up to 31% improvement in the 99% tail latency of the highest QoS class as compared to an uncoordinated operation of the hosts. This result holds in spite of very high burstiness of the traffic.

4.2 PLM Mechanism And Its Deficiencies

4.2.1 SSD Background Activities

SSDs internally pack a lot of complexity and require an internal processor to perform management activities. The management is usually implemented in firmware and is known as *Flash Translation Layer* (FTL) Eshghi and Micheloni (2013), Micheloni and Crippa (2017). The primary role of FTL is to hide the complexities associated with using flash technology and make SSD look like a HDD.

The key characteristic of the flash technology is any programmed bits cannot be overwritten (or updated) until the entire SSD block is reset by a very slow "erasure" operation Bednar and Katos (2011). This implies that any updated data must be written "out-of-place" along with suitable update of the mapping tables to reflect the location of the most up to date data. SSDs have a complex internal structural hierarchy consisting of chips, dies, planes, blocks, and pages, with read/write done on a page-level granularity, whereas erasures are always on a per block basis. (A typical page size is 4-16KB, whereas the block size is much larger at 1-8MB, and getting larger as the technology advances.)

Transparently writing the latest versions of a page in a different place requires a distinction between logical (program specified) and physical page and block addresses Ma et al. (2014); Liu et al. (2016a). The FTL must deal with the resulting logical

to physical address translation, another background activity involved in SSD. The address mapping table is generally cached in the FTL processor's battery backed DRAM for quick access, but any changes are updated to the SSD, which again involves out-of-place writes. Thus, inadequate DRAM or significant changes to the address mapping could cause unpredictable delays in accessing the SSD. These delays cannot be mitigated by NVMe1.4's PLM mechanism, and thus not discussed here.

Another critical characteristic of flash technology is the limited *endurance*, i.e., a block can be erased and reused on a limited number of times before it "wears out" and becomes unusable. The error probability in writing a block increases with its age and needs to be monitored to decide when to retire it or start using it for data that is not updated much Abraham et al. (2010). Inaccurate assessment of SSDs can jeopardise SSDs performance and endurance Mohan et al. (2010); Hu et al. (2011); Agrawal et al. (2008). FTL transparently performs "wear-leveling" in its logical to the physical mapping process and may require filling cold data in blocks close to retirement. Wear leveling can occasionally add latency to SSD access, especially when the SSD is getting closer to its rated lifetimeAlsalibi et al. (2018); Kim and Kwak (2019).

Depending on the policies used to manage out-of-place writes, new blocks will need to be allocated to handle a write if the existing block is getting full. Therefore, out-of-place writes may further consolidate the latest version of several pages into a single block to reduce data scattering and free up some blocks (often known as soft garbage collection). A more extensive "hard" garbage collection that systematically looks for blocks with all modified pages may also be done occasionally. The garbage collection procedure often requires locking an entire flash device plane and can be very time-consuming Cai et al. (2017); Chang et al. (2004); Gal and Toledo (2005); Tavakkol et al. (2018b). The actual erasure of garbage blocks could also interfere with other operationsKim et al. (2019)Tavakkol et al. (2018a).

The above-mentioned background activities can extend the access latency's tail from its nominal value (<100 us) into several milliseconds. For example, Fig. 4.1 shows the latency distribution for a recent Intel SSD Int (2020). It is seen that even 99% latency can exceed 1 ms. Another critical characteristic of flash technology is the limited endurance, i.e., a limit on erasure operation performing on a block before becoming error-proneAbraham et al. (2010).

4.2.2 NVMe SSD Structure

SSDs internally involve complex management activities that are performed largely in the background by the firmware known as *Flash Translation Layer* (FTL). The primary role of FTL is to hide the complexities of out-of-place writes, address translation, wear-leveling, and garbage collection(Chang et al., 2004; Eshghi and Micheloni, 2013; Micheloni and Crippa, 2017). These activities and their interference with normal read/write operations can extend the access latency from its nominal value (<100 μs) into several milliseconds(Kim et al., 2019) as illustrated in Fig. 4.1.

The NVM storage model defines several concepts including NVM subsystem, domain, endurance group, NVM-sets and namespaces, as illustrated in Fig 5.1. An *NVM subsystem*

is an integrated collection of one or more NVMe controllers, one or more interface ports and may contain non-volatile storage and hence an SSD can be considered as an NVM subsystem. An NVM subsystem may consist of single or multiple *domain(s)*, which is the smallest indivisible unit that shares states(for example: power state, capacity information). An *endurance group* is a collection of NVM-

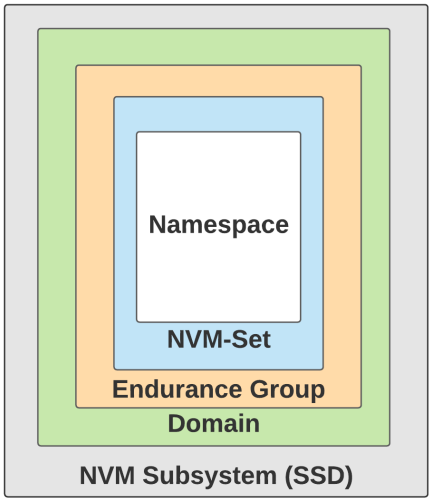


FIGURE 4.2: NVM Storage Hierarchy

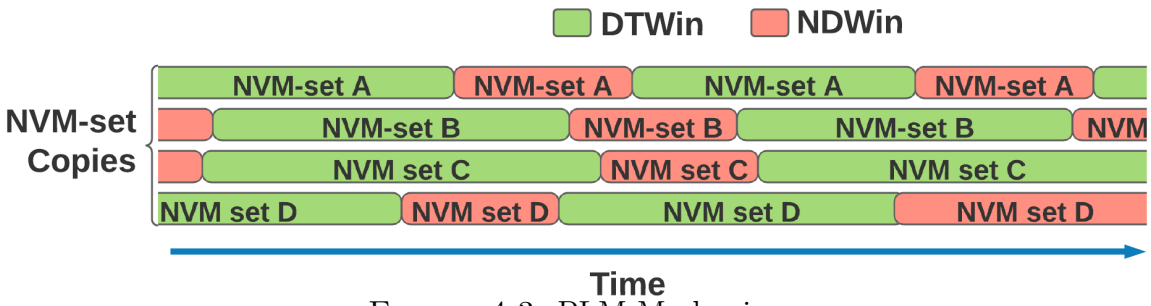


FIGURE 4.3: PLM Mechanism

Sets, which consist of one or an array of namespaces. Each endurance group is an organizational construct for wear leveling purposes.

The *NVMe Storage Controller* (NSC) helps to connect a host with the NVM storage. An SSD can be accessed by a host locally as well as remotely. For remote access, a protocol called NVMe-oF (NVMe over fabric) has been defined that essentially extends the NVMe commands to be ported from host to the target and executed remotely (Minturn and Metz, 2015).

The PLM is enabled at NVM-set granularity and, therefore, NVM-set cycles between the aforementioned DTWin and NDWin time windows as fig 5.2 such that all background activities are concentrated only during NDWin period. The **DTWin budget** is defined by two attributes: (a) a predefined time limit and (b) a predefined limit on the number of the read and write operations that can be performed on an NVM-set. The NVM-set may transition to NDWin if either of these limits is exceeded. There are various DTWin attributes introduced in NVMe v1.4, presented in table 4.1, such as DTWin read typical (C_{RD}), DTWin Write typical (C_{WR}), DTWin time maximum (T_{DW}). We consider the parameters C_{RD} and C_{WR} as DTWin "counts" (DC), and DC/T_{DW} as the DTWin budgets.

Table 4.1: DTWin attributes involved in PLM

Attribute Name	Symbol	Definition
DTWin Reads Typical	C_{RD}	Indicates the typical number of 4 KiB random reads that may be performed in the Deterministic Window
DTWin Writes Typical	C_{WR}	Indicates the typical number of writes in units of the Optimal Write Size that may be performed in the Deterministic Window
DTWin Time Maximum	T_{DW}	Indicates the maximum time in milliseconds that the NVM-set is able to remain in a Deterministic Window before entering a Non-Deterministic Window
NDWIN Time Minimum High	T_{NDWH}	Indicates the minimum time in milliseconds that the NVM-set needs to remain in the Non-Deterministic Window before entering a Deterministic Window. This is the time necessary to prepare for remaining in the Deterministic Window for DTWin Time Maximum
NDWIN Time Minimum Low	T_{NDWL}	Indicates the minimum time in milliseconds that the NVM-set needs to remain in the Non-Deterministic Window before entering a Deterministic Window. This is regardless of the amount of time spent in the previous Deterministic Window
DTWin Reads Estimate	E_{RD}	Indicates a reliable estimate of the number of 4KiB random reads remaining in the current Deterministic Window, if applicable. This value decrements from DTWin Reads Typical to 0h based on host read activity and operating conditions
DTWin Writes Estimate	E_{WR}	Indicates a reliable estimate of the number of writes in units of the Optimal Write Size remaining in the current Deterministic Window, if applicable. This value decrements from DTWin Writes Typical to 0h based on host write activity and operating conditions
DTWin Time Estimate	E_T	Indicates a reliable estimate of the time in milliseconds remaining in the current Deterministic Window

4.2.3 PLM Feature And Its Functioning

The PLM feature is currently designed primarily to cater to reads since the latencies for reads are generally more critical than for writes. To achieve the deterministic latency, the host needs to obey the predictable latency operating rules. As stated earlier, to provide continuous access to DTWin mode to an application, we need multiple (two or more) different NVM-sets that are identical and contain copies of the data. If all NVM-set copies are never simultaneously placed in the NDWin state, an application can read data in the DTWin mode any time. While the required duplication is expensive, it is needed only for the data accesses that must

be deterministic; one copy suffices for all other data. Also, if multiple copies are maintained for resilience reasons, they can be used to provide the deterministic service as well. It is also worth noting that a traditional RAID1 (mirroring) does not provide the same functionality as PLM, since the reads could occur while the SSD is doing the background operations.

The NVM-set may transition to the NDWin before exhausting its DTWin budget – this happens if there is an emergency management operation required such as an emergency garbage collection being triggered. The amount of time spent by the NVM-set in NDWin state depends on the necessary background operations for management activity.

A remote host can setup a connection via NSC. The NSC advertises the DTWin related status for each NVM-set into a "log-page". The log-page is accessible to the host using NVMe commands and therefore the DTWin information per NVM-set. The host defines the beginning of DTWin and NDWin periods by making requests to the NSC using "set-feature" command. Since a host could use multiple NVM-sets simultaneously, they could be at different points in their DTWin/NDWin cycles. Therefore, the host must influence DTWin/NDWin duration to synchronize multiple copies of NVM-sets. PLM's current specification puts the regular transition between DTWin and NDWin control with the host (if the log-page advertised status are respected) and the forced transition with the NSC (if the host does not respect operating rules).

4.2.4 Issues With PLM Feature In A Multi-host Environment

We now study the issues of the PLM feature in a multi-host environment. To illustrate this, we have created a mixture of three different workloads of different QoS classes (i.e. high, medium, and low) considering a few IO-intensive portions from Systor 2017 traces (Repository, 2017), a month-long virtual desktop infrastructure (VDI)

read-intensive trace. The read IO size of all three considered workloads, which shows a wide variations, have studied the impact on latency for high QoS workloads in a shared environment when multiple workloads share the same data storage resources without the awareness of others.

The impact on 99.99 percentile tail latency per DTWin for high QoS workloads in the said workload mixture is represented in Fig. 4.4. We observed that the high QoS workload violated the latency requirement. The high QoS workload experienced 6X times more 99.99 percentile tail latency when running with other QoS class workloads, in a span of 15 minutes, compare to running in isolation. Thus, it gives us the insight to engage a PLMC that schedules IO requests from different workloads during a deterministic period such that the QoS requirement is satisfied.

4.3 Providing Deterministic Service In Multi-host Environment

The current PLM feature does not recognize a multi-host environment; instead, it assumes that each host will independently work with the PLM feature of the target NVMe SSD device that it is trying to use. In a single host environment, a host can access an NVM-set during DTWin via NSC. In the shared environment a DC allocate module is necessary along with traditional NSC functionalities. This module can help to distribute the required DC for different workloads running on a single host. However, running separate workloads in an environment of multi-host systems might create a bottleneck at the NSC if the DC-allocate module is placed as a part of NSC. Therefore, the DC-allocate module needs to be outside of the NSC to coordinate among different host accesses to control the access latency tail in a multi-host environment. Hence, storage server could be suitable place for PLMC. However, it is also possible to locate it at the host side, acting as one of the requesting hosts for the shared NVMe set. For fault-tolerance purposes, the PLM host can be dynamic using standard leader election algorithms (Gharehchopogh and Arjang, 2014), which

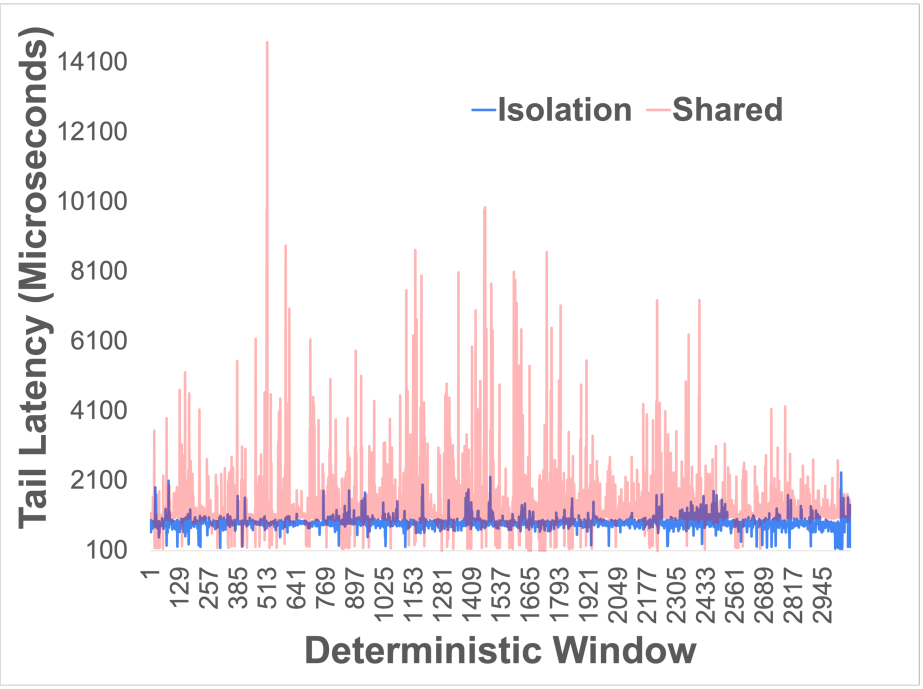


FIGURE 4.4: 99.99 percentile Tail Latency per DTWin in isolation vs sharing

we do not focus upon In this work.

4.3.1 Coordinating Multiple Host Accesses Through PLMC

The proposed PLMC is shown in Fig. 4.5. The PLMC allocates DC to each of the hosts actively accessing the NVMe target, consisting of one or more NVM subsystem(s), i.e SSD(s). Each subsystem can have one or more NSCs that control a number of NVM-sets. The PLMC uses the conventional NVMe-OF (NVMe over fabric) protocol (Minturn and Metz, 2015; Guz et al., 2017) to communicates with NSC(s) and access each NVM-set's log-page to collect DTWin attributes. The PLMC is not involved in the data path for scalability reasons and does not directly monitor IOs. Instead, each host is trusted to accurately convey its usage to the PLMC and abide by its limits. On its end, the PLMC should learn the needs of the workloads run by each host and supply the DC accordingly. Therefore, the active hosts can communicate with PLMC via message passing to provide for their needs. The host,

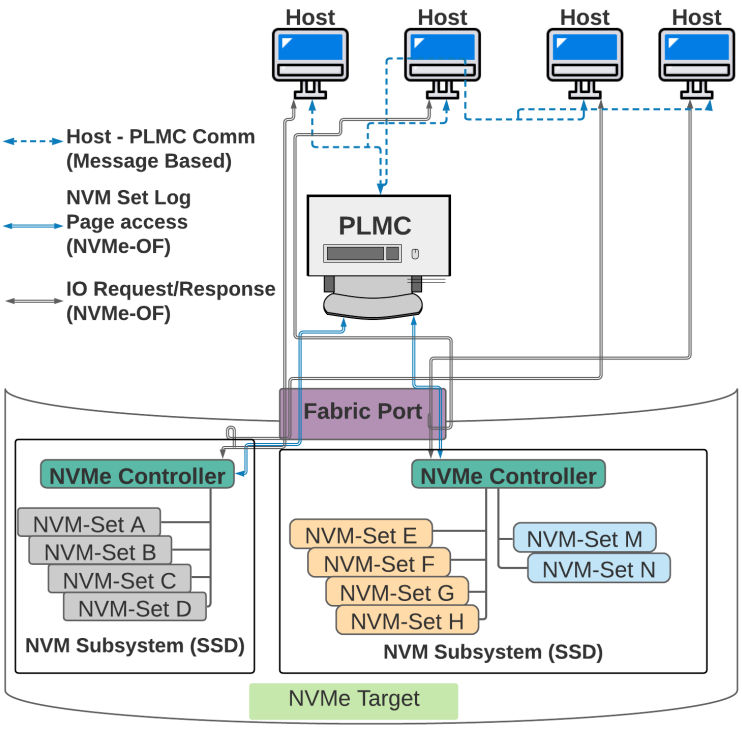


FIGURE 4.5: Proposed PLMC Architecture

with the allocated DC, can perform remote IO operations via the NVMe-oF protocol.

All hosts with similar traffic characteristics will experience the same underlying device access latency (exclusive of any queuing or other latencies) in the absence of management operations; therefore, the difference will be mostly in tail latency. The end-to-end latency will consist of at least four components: (a) host-side IO dispatching and IO completion latency, (b) network transit latency, (c) NVMe queuing and queue handling latency, and (d) device access latency. QoS classes' definition is most meaningful in terms of the end-to-end latency, and thus control over the overall tail latency needs to consider all these four components. It would require decomposing the end-to-end tail latency into its constituent elements and managing each part suitably. However, this paper is only focused on (d).

The PLMC considers the traffic characteristics experienced by each of the hosts and the available DC of an NVM-set to distribute the DC among hosts. Moreover, an efficient DC allocation mechanism should consider the QoS class requirements.

PLMC supports multiple application classes based on the tail latency requirement. To obtain deterministic latency, a host needs to access the device during DTWin while respecting DTWin attribute values. Therefore a suitable DC allocation policy is needed. Unlike a static DC allocation we have considered a static initial DC allocation along with dynamic additional DC allocation in PLMC. We assume that each host requests allocation of DC from PLMC at the start of DTWin to have the initial allocated DC. If the host receives fewer than the required counts during DTWin, it can request PLMC for additional DC allocation. The allocated during a DTWin are not taken back. Therefore, an appropriate DC allocation policy is required to reduce the unused DC during a DTWin. PLMC explores various DC allocation policies In this work.

4.3.2 Host Traffic Estimation By PLMC

The PLMC's primary role is to estimate or receive each of the participating hosts' DC needs and make the allocation. As stated earlier, PLMC residing locally at the storage server and can register itself with the device as another NVMe host. It is also possible to locate the PLMC at the host side, acting as one of the requesting hosts for the shared NVM-set. For fault-tolerance purposes, the PLM host can be dynamic using standard leader election algorithms Gharehchopogh and Arjang (2014), which we do not focus upon In this work.

The critical parameter required by PLMC is the estimate of the DC allocated to each host for the next DTWin. We have found the storage traffic to be generally quite irregular and nonstationary. Since the well-known time-series prediction methods such as Kalman Filter, ARMA models, etc. assume at least quasi-stationarity, they were not found to be useful, and we used a simple exponential smoothing-based prediction. We will comment on this aspect in section 5.4.

Let $R_j^{(m)}(n)$ denote the measured request count (traffic) at the n-th scheduling

period for the j-th class and $R_j^{(p)}(n)$ their smoothed estimated in the same period, with $0 < \zeta < 1$ as the smoothing constant. Then,

$$R_j^{(p)}(n+1) = \zeta R_j^{(m)}(n) + (1-\zeta) R_j^{(p)}(n) \tag{4.1}$$

The DC allocations are done before the scheduling window starts and do not change during the window. We assume that each host specifies some minimum DC requirement, and any excess remaining device DC's are split up among the hosts based on some policies, two of which are discussed in the following.

4.3.3 DC Allocation

As stated above, PLMC uses a simple exponential smoothing based estimation of the DC needs of various classes, which we call *Coordinator Initiated Prediction*. It is also possible to perform a *host-based prediction* and convey those to PLMC. The hosts may have better estimates of the traffic, and thus their estimation may be preferred. However, the quality of assessment may vary. For our trace-based experiments, we can count precisely how many requests will happen in the next window and supply that to PLMC. Thus, host-based prediction's sole purpose is to use it as a baseline to determine how much better the allocation policy can do if we knew the DC requirements exactly in every DTWin period. We consider the following two policies for allocating the available DC among different classes:

- Policy I (Strict Priority): Here we first estimate the minimum average DC for each host, say M_i for the i-th host. Let $M = \sum_i M_i$ denote the total minimum DC for all k hosts and D the total count for the device, which is as same as C_{RD}. If $M < D$, then each host can be assigned M_i DCs, for $i = 1, 2, ..., k$. If $M > D$, we use bin packing, assuring that each host will be allocated at least their minimum DC required. Therefore, the allocation starts with the estimated count for the high priority until we run out of counts.

Table 4.2: Table of notations

Symbol	Explanation
$W(n)$	Actual duration of nth scheduling Window.
$W_d(n)$	Actual duration of nth DTWin, which can be less than or equal to T_{DW} at nth window
$W_{nd}(n)$	Actual duration of nth NDWin such that $T_{NDWL} \leqslant W_{nd}(n) \leqslant T_{NDWL}$
C	Available DC for the device, which is as same as C_{RD}
$C^{\text{res}}(n)$	Residual DCs after minimum allocation to all classes during $W_d(n)$
$C_j^{\min}$	Required minimum DC for j-th class
$C_j^{\text{used}}(n)$	Consumed DC for j-th class during $W_d(n)$
$C_j^{\text{alloc}}(n)$	Allocated DC for j-th class during $W_d(n)$
$D_j^{\min}$	Minimum required allocation for class j
$\phi_j(n)$	Allocation of excess DC to class j (under strict priority or fixed ratio policies)
$L_j^{(m)}(n)$	Tail Latency measured for j-th class during $W(n)$
L_d	Latency of 4KB IO in a DTWin
L_{nd}	Latency of 4KB IO in a NDWin
r_j	Fixed ratio for j-th class to assign the residue DC
$R_j^{(m)}(n)$	Measured Traffic (#IOs/DTWin) for j-th class during $W(n)$
$R_j^{(p)}(n)$	Predicted Traffic (#IOs/DTWin) for j-th class during $W(n)$
T_{DW}	Maximum DTWin Timeperiod
$u_j(n)$	Allocated DC utilization for j-th class during $W(n)$
$v_j(n)$	Deficiency of DC during $W(n)$

- Policy II (Fixed Ratio): The minimum average DC is computed as Policy I and can have two different cases. In case the sum of the minimum DC less than D, we deal similarly as earlier. In the other case, each of the hosts is assigned the minimum required DC. The remaining DCs are allocated to each of the hosts based on the QoS-based ratio. For three QoS classes, High, medium, and low, the medium is assigned double as much as low QoS, and the high is given double as much as medium.

4.3.4 Analysis Of DC Allocations

Consider k QoS classes for applications, denoted as $Q_1, Q_2, ..., Q_k$, with tail latency requirement of L_i for class Q_i. Also consider p identical NVM-sets $S_1, S_2, ..., S_p$ in a NVM-set pool. In the following we use the index n to denote the n-th window DTWin/NDWin. The n-th DTWin is of duration $W_d(n)$, followed by a NDWin of

duration $W_{nd}(n)$. We define the total duration of these as the *scheduling period* and denote it as $W(n)$. Obviously,

$$W(n) = W_d(n) + W_{nd}(n), \text{ for } m = 0, 1, ... \tag{4.2}$$

The complete set of notations are given in Table 4.2. The maximum duration of $W_d(n)$ is $T_{DW}(n)$ – the specified period (100ms or 400ms in our experiments); however, if all of the available DCs are exhausted early, i.e. $W_d(n) \leqslant T_{DW}$, the window also ends prematurely. The premature end of the window would include the service of all the requests arriving during the window. The duration of the NDWin period depends on the write request behavior during the preceding DTWin.

In each scheduling period, we need to distribute the total DC, C (as same as C_{RD}), among the k classes (applications) such that the corresponding target latency requirement is met and no counts are wasted (i.e., reserved for a class that does not use them). In particular, the allocation of DCs should consider the following three aspects:

Definition 1 (Utilization)**.** The utilization of a host is defined by the ratio between the number of DCs consumed vs. allocated.

Definition 2 (Deficiency)**.** The deficiency of a host is the fraction of requests that are not covered by the allocated DC.

Definition 3 (Tail Latency)**.** The 99.99-percentile tail latency is measured for each QoS class considered.

The DC allocation for each host during a given $W_d(n)$ is as follows. If the total predicted traffic $\sum_{j=1}^{q} R_j^{(p)}(n)$ is below C, then we allocate the predicted value of DC. Otherwise, the residual DC $C^{res}(n)$ (i.e., the counts remaining after the minimum allocation during $W_d(n)$) can be distributed among the q hosts based on their QoS class. We consider the following two methods for this:

1. **Policy I (Strict Priority):** This additional allocation policy prioritize the high priority DC needs. The high priority workload get priority to assign the required DC to match the predicted traffic for the current window, if residual DC is available. If the residual DC is less than the predicted traffic, all of it is allocated to that traffic.
2. **Policy II Fixed Ratio:** In this additional allocation policy, the residual DC is split among various hosts based on a predefined set of ratios r_j corresponding to the different classes of workloads running.

4.3.5 Deterministic Count Allocations By PLMC

We call the exponential smoothing-based estimation of DC as *Coordinator Initiated Prediction*. It is also possible to make a *host-based prediction* and convey those to PLMC. The hosts may have better estimates of the traffic, and thus their estimation may be preferred. However, the quality of the estimation may vary. We can incorporate individual traffic predictor at each host to precisely estimate the number of requests happen in the DTWin and supply that to PLMC. Thus, the sole purpose of host-based prediction is to use it as a baseline to determine how much better the DC allocation policy can do if we knew the exact DC requirements in every DTWin period.

Let us consider k hosts each running workloads of different QoS classes, denoted as $Q_1, Q_2, ..., Q_k$, with tail latency requirement of L_i for class Q_i. Also, let's consider p copies of NVM-sets $S_1, S_2, ..., S_p$. In the following we use the index n to denote the n-th window DT/ND window. The n-th DTWin is of duration $W_d(n)$, followed by an NDWin of duration $W_{nd}(n)$. We define the total duration of these as the *scheduling period* $W(n) = W_d(n) + W_{nd}(n)$.

The maximum duration of $W_d(n)$ is $T_{DW}(n)$ – the specified period (100ms or 400ms in our experiments); however, if all of the available DCs are exhausted early,

i.e. $W_d(n) \leqslant T_{DW}$, the window also ends prematurely. The premature end of the window would include the service of all the requests arriving during the window. The duration of the non-deterministic period depends on the write request behavior during the preceding DTWin.

In each scheduling period, we need to distribute the total DC C (as same as C_{RD}) among the k hosts such that the corresponding target latency requirement is met and no counts are wasted (i.e., reserved for a host that does not use them). In particular, the allocation of DCs should consider the following three aspects and use them as evaluation metrices.

Definition 4 (Utilization)**.** The utilization of a host is defined by the ratio between the number of DCs consumed vs. allocated.

Definition 5 (Deficiency)**.** The deficiency of a host is the fraction of requests that are not covered by the allocated DC.

Definition 6 (Tail Latency)**.** The 99.99-percentile tail latency is measured for each QoS class (same as host In this work).

The DC allocation for each host during a given $W_d(n)$ is as follows. If the total predicted traffic $\sum_{j=1}^{q} R_j^{(p)}(n)$ is below C, then we allocate the predicted value of DC. Otherwise, the residual DC $C^{\text{res}}(n)$ (i.e., the counts remaining after the minimum allocation during $W_d(n)$) can be distributed among the q hosts based on their QoS class. We consider the following two methods for this:

1. **Policy I (Strict Priority):** This additional allocation policy prioritize the high priority DC needs. The high priority workload get priority to assign the required DC to match the predicted traffic for the current window, if residual DC is available. If the residual DC is less than the predicted traffic, all of it is allocated to that traffic.

2. **Policy II Fixed Ratio:** In this additional allocation policy, the residual DC is split among various hosts based on a predefined set of ratios r_j corresponding to the different classes of workloads running.

4.4 Evaluation Of The Proposed PLMC

Since the PLM feature is quite new, currently there are no commercially available SSDs that support it. Consequently, *the only direct way to evaluate PLM capability is to use a comprehensive simulation model of SSDs supporting PLM*. For this we built the evaluation capabilities around the existing MQSim SSD simulation package as described below.

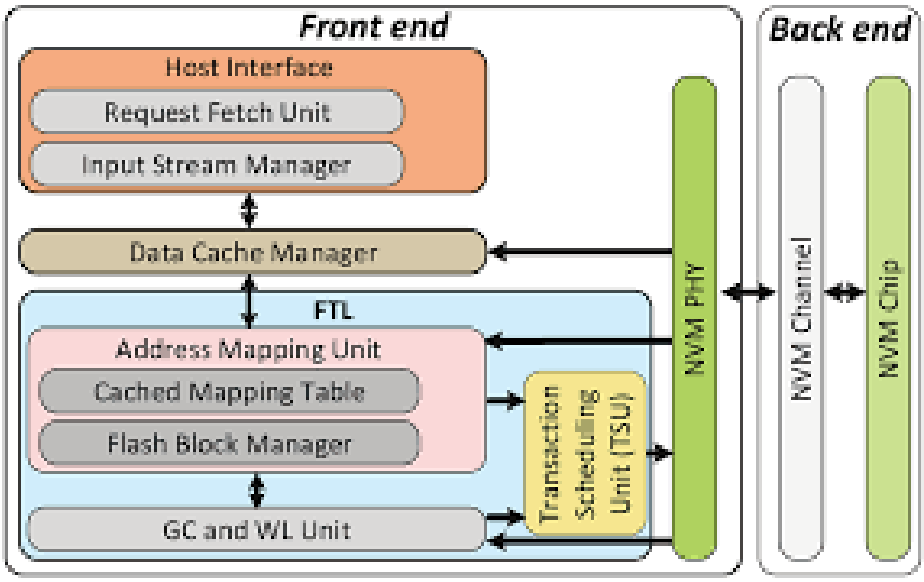


FIGURE 4.6: High-level view of MQSim components

4.4.1 MQSim SSD Simulator

MQSim Tavakkol et al. (2018b) attempts to build a very detailed and realistic model of SSDs. It explicitly represents the flash device characteristics and operation, internal architecture of the SSD, detailed FTL operations, device level caching, and host

interfacing. It provides a detailed representation of both SATA and NVMe interfaces, of which we use NVMe differentiated queuing feature to support different QoS classes.

Fig. 4.6 shows the organization of MQSim. MQSim simulates nearly all aspects of SSD structure (chips, dies, planes, blocks, pages), buses (channels and inter-chip/die buses), and most non deterministic background operations associated with nonvolatile media (logical to physical address mapping, page reads, out-of-place page updates, consolidation of pages, garbage collection, block erasure, wear leveling, etc.). The developers have validated its performance against several real SSDs and found them to be very close, with the response time errors of 11% (average) and 18% maximum.

However, MQSim has some limitations and does not contain all the features necessary for building the PLM. It does not support the concept of NVM-sets or other newly proposed NVMe 1.4 features. Consequently, we embarked upon an extensive effort to understand the implementation and enhance it for our needs, as explained below.

4.4.2 Enhancing MQSim Simulator To Support PLMC

In the absence of NVM-sets, PLM emulation would require multiple SSDs, but MQSim simulates only a single SSD. It also does not support multiple hosts. For the latter, we assumed that each IO flow defined in MQSim is a host for our purposes. We emulated multiple devices by virtualizing the entire address space into logical address ranges pertaining to the the number of required devices. MQSim also assumes that the logical address space is divided amongst concurrently running IO flows (i.e each flow accesses its own address range). We changed this aspect so that the requests from all flows are served from the same address range.

Proceeding to the PLM features, we first had to introduce the notion of IO Determinism by stalling maintenance activities and carry them out at a later time (during NDWIN period). MQSim has two triggers for garbage collection (GC): (a)

soft trigger, that periodically looks for blocks with very few valid pages and maps those pages out, thereby enabling the block for erasure, and (b) hard trigger, which is initiated when the number of clean blocks falls below some limit. The soft-trigger GC can be postponed without any ill effects, but the hard-trigger cannot. We switched off the soft trigger GC so that no GC takes place during the normal (and supposedly DTWin) period and is only triggered during the NDWin period on command. The hard trigger GC remains untouched as it is intended for the NDWin period.

We also switched on a configurable parameter in MQSim that ensures that the address mapping table is cached in the SSD DRAM. As stated in 5.2A, misses in the mapping table can be quite expensive and cannot be moderated by the PLM. While this caching protects the DTWin period, it does the same for NDWin. With limited caching, it would be desirable to intelligently prefetch (and evict) map table entries during the NDWin period so that very few misses are experienced during the DT period, but that's extremely difficult to ensure in reality.

For the DTWin, NDWin and DT Count features, we introduced some parameters that can be set at the start of a simulation run. These include: (a) Maximum DTWin and NDWin time period (in nanoseconds) after which the SSD will autonomously transition to the other state, and (b) DTCount value, from which we allocate counts to all the hosts. MQSim also does not have provision to measure the tail latency, and we have included reporting of 90th percentile tail latency experienced by the host when the DTWin period ends or when it exhausts all allocated DT counts during the given window).

Simulating NDWin period with actual maintenance operations (e.g., garbage collection, wear leveling, block consolidation, erasure) is difficult in MQSim; therefore, we emulate unpredictable delays during NDWin period by introducing additional latency along with the soft GC which may be triggered during the NDWin period. This delay is assumed to follow an exponential distribution.

4.4.3 Workloads And Configurations Used

For evaluation, we used the Systor 2017 trace (Repository, 2017), a month-long virtual desktop infrastructure (VDI) read-intensive trace that exhibits wide variations in IO sizes.

We combined selected portions of the original trace to create a variety of workloads listed in Table 4.3 (tabulated as average incoming 4KB requests during each DTWin). Each of these datasets consist of three workloads with different IO intensities, considered as high, medium and low QoS classes respectively. We have considered these priorities based on their 99.99 percentile tail latency requirement, where high has lowest latency requirement among all others.

Table 4.3: Datasets' Read IO Intensities

DataSet	High	Medium	Low
DS_1	420	0.78×High	0.71×High
DS_2	480	0.83×High	0.72×High
DS_3	165	0.85×High	0.7×High

We evaluated several different configurations (w.r.t DC and DTWin period length of the NVM-set) for each of these datasets. A large DTWin reduces interactions between the hosts and PLMC, reduces overhead, and may reduce burstiness due to the aggregation effect. However, since DCs are readjusted only once for each DTWin, a large DTWin will reduce the effectiveness of the control.

Table 4.4: Configurations used

Config	DC	T_{DW}
1	$1.2 \times M_{DC}$	100 ms
2	$1.4 \times M_{DC}$	100 ms
3	$2.0 \times M_{DC}$	400 ms
4	$2.5 \times M_{DC}$	400 ms

Because of this trade-off, we have used DC and DTWin period (T_{DW}) configurations as represented in Table 4.4. Here M_{DC} is the average number of DC required by all three workloads, which has been determined offline by analyzing the trace. In all cases, the available DC of an NVM-set at any DTWin period is set above the average value to handle high traffic variability.

Config	Policy	Utilization			Deficiency					
		10 percentile			50 Percentile			90 percentile		
		High	Med	Low	High	Med	Low	High	Med	Low
2	SP	0.56	0.56	0.56	0	0.03	0.05	0.28	0.35	0.33
	FR	0.54	0.56	0.56	0	0.11	0.11	0.15	0.35	0.36
4	SP	0.87	0.88	0.89	0	0.02	0.04	0.28	0.49	0.49
	FR	0.84	0.88	0.89	0	0.27	0.28	0.15	0.54	0.54

FIGURE 4.7: Evaluation result for CoD based 10 percentile Utilization, 50 and 90 percentile Deficiency considering DS1.

4.4.4 Evaluation Results

For evaluation, we considered three hosts, each running workloads of high, medium and low QoS classes respectively. We evaluate how DC of two NVM-set copies (or two SSDs) are distributed amongst the hosts based on DC utilization, DC deficiency, and resulting 99.99% tail latency respectively.

The results obtained from Datasets 1, 2 and 3 are similar and therefore only Dataset1 results are reported. The three aforementioned QoS class workloads are considered for both strict priority and fixed ratio policies under coordinator directed (CoD) and host-directed (HoD) cases.

Utilization: In Fig. 4.7 we only report 10 percentile Utilization values for the CoD case (w.r.t. both strict priority and fixed ratio) as it was observed that utilization is always 100% for HoD (because the exact amount of DC needed is known). The 50 & 90 percentiles are not reported since they all turn out to be 100%, which indicates no wastage of allocated counts and may be a sign of scarce resources. The CoD numbers are lower than the HoD numbers because the PLMC does not know the precise needs and therefore has to be generous in its estimation. Thus the excess DT count allocation results in less than 100% utilization in Fig 4.7. All three hosts experience roughly the same utilization level, with the "Low" dominating slightly. The reason behind this is "Low" QoS host is allocated less DC compared to the

FIGURE 4.8: 99.99 Percentile Tail Latency for HoD Simulation

FIGURE 4.9: 99.99 Percentile Tail Latency for CoD Simulation

traffic experienced and hence, uses up all allocated DC.

Deficiency: For deficiency, we report the 50 & 90 percentile values for the CoD case in Fig 4.7. It is observed that the median deficiencies are rather small compared to the 90 percentile deficiency value. For HoD we observed that the deficiencies are closer to zero as number of required DC are correctly assigned. However, for CoD, they are small too since the available DCs are significantly higher than the average. The reported 90 percentile deficiency numbers are much more variable. It is worth noting that the experiment runs for ≈9000 DTWins for configuration 2 and 2250 for configuration 4, which indicates that the 90 percentile values are expected to be rather variable.

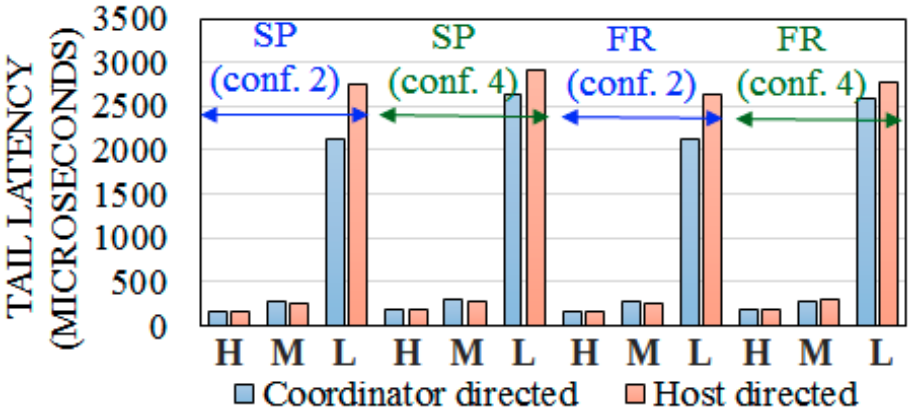


FIGURE 4.10: Comparison of Median tail latencies for Dataset 1.

Tail Latency: For tail latency, we report 99.99 percentile values in Fig. 4.8 and 4.9 for HoD and CoD respectively. We compute the the tail latency during each DTWin and then report the top 99.99% values experienced during the entire workload lifetime for each "High", "Medium" and "Low" QoS traffic. For both CoD and HoD, High QoS class achieves smallest 99.99 percentile tail latency as compared to Medium and Low. Both HoD and CoD improve the 99.99 percentile latency for high 87.2% and 81.7%, medium 56.9% and 42.9%, low 42.5% and 16.2% respectively, when we consider strict priority with Config2. Using the strict priority and fixed ratio the High QoS class CoD attains 30% and 25% higher than 99.99 percentile tail latency of HoD. Such accuracy is impressive, considering the stress case considered here: very high burstiness and synchronized traffic with simple traffic estimation. Moreover, the High QoS class in CoD for strict priority can improve 93.2% compare to the shared environment(Fig. 4.4).

We next simulate a less challenging situation where we create another dataset (henceforth called Dataset4) from Dataset1 by adding an offset of 200 ms and 400 ms to "Medium" and "Low" traffic, respectively. This change desynchronizes the peaks and results in a traffic pattern where satisfying the requirements of all classes becomes much easier. Fig. 4.11 shows the variation of tail latency for Dataset4. It is seen that

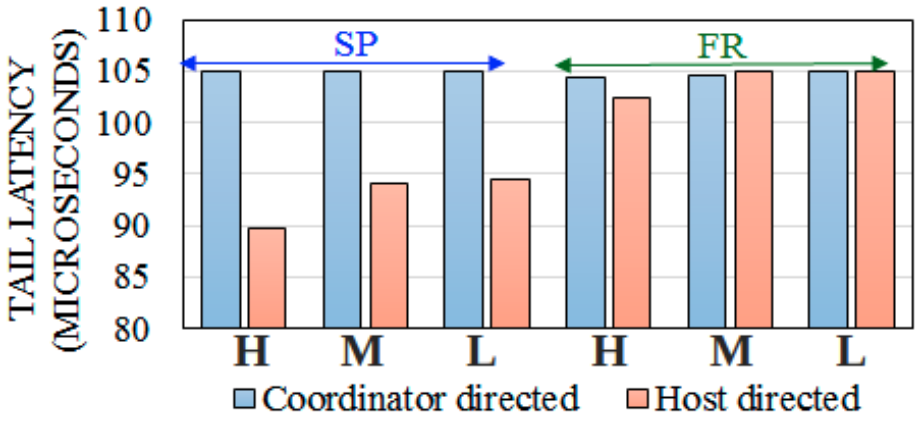


FIGURE 4.11: Comparison of Median tail latencies for Dataset 4.

the latency drops drastically, by almost 30X, as compared to Fig. 4.10. With such low latencies, the QoS priority issue is moot, and all hosts get essentially the same treatment. The agreement between HoD and CoD is excellent for fixed ratio and very good for strict priority (16% error for high priority). However, any discrepancy, in this case, is irrelevant since all QoS objectives achieved.

4.5 Conclusions And Future Work

This paper explores and extends the predictable latency mode (PLM) feature introduced in NVMe v1.4. We show how the PLM concept can be extended to a multi-host environment where multiple hosts share an NVMe device, as would be the case for database accesses. For this, we have proposed PLM coordinator (PLMC) that runs on a storage server and can be implemented without any changes to the current NVMe v1.4 standard. We evaluated the proposed PLMC with extensive experiments performed in SSD simulator (MQSim). The experimental evaluation of the mechanism demonstrates that it can substantially reduce tail latency and help deliver predictable latency to the higher QoS classes with 93.2% improvement in 99.99 percentile tail latency compared to the shared environment. In future, we will also examine the impact of writes regarding their effect on the duration of the

nondeterministic window and their impact on the deterministic mode operation by filling up the NVRAM buffer and forcing a switch-over to the nondeterministic mode.

CHAPTER 5

ACHIEVING DETERMINISTIC LATENCY IN PRESENCE OF WRITE-INTENSIVE WORKLOAD

5.1 Introduction

Storage systems form a crucial component of data center infrastructure and substantially influence the performance of the applications. Furthermore, nearly all of the storage resides away from the hosts in "storage servers," which means that network is an indispensable part of any storage access in current days. The emerging interactive applications are increasingly data-centric and require a low average storage access latency and one that does not vary a lot from request to request (Xu et al., 2013; DeCandia et al., 2007; Dean and Barroso, 2013). The low variation in latency is popularly expressed as a requirement of "deterministic" latency, but it means that the length and mass in the latency tail are tightly controlled.

The enterprise's ongoing replacement of hard drives (HDDs) by NVMe based solid-state drives (SSDs) has enabled high storage throughput and low storage access latency. Unfortunately, SSDs also need to internally carry out a complex array of background activities to create the impression of a simple read/write device. This simplicity is achieved by the Flash Translation Layer (FTL) that manages out-of-place

writes, address translation, wear-leveling, and garbage collection (Alsalibi et al., 2018; Cai et al., 2017; Tavakkol et al., 2018b; Kim et al., 2019; Tavakkol et al., 2018a). These activities can occasionally increase the IO latencies substantially, even to the level of several milliseconds (Int, 2020).

Traditionally, the SSD firmware controls these background activities with little regard to the ongoing IO operations. Certain implementation choices such as incremental garbage collection or incremental wear-leveling can reduce the disruptive impact of these operations (Huang and Chang, 2018; Guo et al., 2017; Du et al., 2017; Do et al., 2019). However, to eliminate them, it is necessary to either put the scheduling in the host's hands (so that the host can schedule them as appropriate) or provide a mechanism by which the applications needing determinism can always have their requests processed without interference. The former is supported by the emerging concept of ZNS (Zoned Name Space) (NVM, 2019) where the SSD is divided into several "zones" each of which is managed independently as a log-structured "device," i.e., data can only be appended until the zone fills up. The host controls the garbage collection of zones and can be avoided during IOs of applications requiring deterministic latency. However, the additional burden placed on the host for managing zones is a downside, making its widespread adoption difficult.

NVMe 1.4 specification (NVMe v1.4) proposes another mechanism to achieve deterministic latency without burdening the hosts. However, it has the disadvantage of requiring the data to be replicated on two (or more) identical devices, similar to but not identical to RAID 1. It is known as the Predictable Latency Mode (PLM), and each SSD in the RAID-1 like pair cycles through time windows known as "DTWin" (deterministic window) and "NDWin" (non-deterministic window) in a way that at least one of them is always in the DTWin mode. In the DTWin mode, background operations are held back so that the device is dedicated to serving the requests.

NVMe1.4 standard does not offer clear guidance as to how write I/O should be

handled. This standard suggests that the writes during the DTWin period could be absorbed by an NVRAM buffer inside the SSD and then transferred to SSD during the NDWin period. Such an arrangement can work nicely for read-only traffic since the data can be read from whichever SSD is in DTWin mode; however, writes introduce three complications: (a) the writes must be reflected on both copies, (b) writes could trigger background activities (e.g., garbage collection) and (c) a write cannot be postponed as that would result in serving stale data. However, this brings in several issues regarding the sizing of NVRAM buffer and corresponding cost implications and the impact of buffer overflow (which would require a forced transition from DTWin to NDWin mode). The NVMe1.4 mechanism does not address the issue of data being shared among multiple hosts, such as in the case of a distributed database system. Moreover, no commercially available SSD has implemented this feature so far; therefore, it is impossible to evaluate it directly. In the work (Roy et al., 2021), the read IO management is explored in a shared environment by modifying an extremely detailed SSD simulator called MQSim (Tavakkol et al., 2018b). The designed coordinator synchronizes hosts (that itself looks like a host to the device) to observe the IO use of each host and distributes the IO capability during the DTwin period among various hosts. We also introduced the notion of QoS and showed how the IO count allocation could be done to respect the QoS requirements.

Although we continue with a similar conceptual model, we take a radically different approach In this work. **First**, this paper attempts to examine the issue of determinism in an actual implementation rather than via simulation. **Second**, because of the unavailability of an actual device that provides the PLM feature and the vast storage infrastructure based on regular SSDs, we propose to use only a regular SSD for implementing the PLM features. This modified architecture is named as PLM$^{\text{light}}$In this work. Although the background operations may occur at random times in a regular SSD, we attempt to reduce their impact by bunching up the writes and not

doing them during DTWin period. (The DTWin/NDWin periods are now artificially imposed durations explained later). **Third**, we exploit the emerging nonvolatile memory technology with rather deterministic access times (i.e., Intel Optane SSD considered) to buffer the writes during the DTWin, and eventually flush them to the SSD en-masse during the NDWin period. The write flush to SSD may trigger some garbage collection activity, which we expect to happen in the NDWin period. The objective then is to examine to what extent we can achieve deterministic read latencies and low write latencies with this arrangement.

We still focus on the operation in a multi-host environment where it becomes necessary to coordinate hosts' access so that each host stream can be given the desired QoS treatment. In particular, we assume that each host's IO stream belongs to one of the predefined set of QoS classes, and the coordinator, which we call PLM$^{\text{light}}$Coordinator (PLMLC), assigns how much of the IO capacity of a DTWin period each class can use. Any excess IO operations must be done in the NDWin period, where the latency is no longer deterministic. We show that PLMLC can help in achieving low latency and provides QoS differentiation as well. Although the overall performance is not as good as can be achieved via a real implementation of PLM, we note that we can achieve these results with widely available technologies and thus the solution can be easily used by existing storage systems without any need to replace the existing devices.

5.2 Background

5.2.1 SSD Structure and Management Activities

Fig.5.1 shows the details of NVM storage structure. An NVMe SSD is partitioned into one or more "NVM subsystems" which can be managed independently. An NVM subsystem comprises of one or more NVMe controllers, one or more NVM subsystem ports, non-volatile storage medium and an interface between the controllers and the

storage medium. Therefore, an SSD can be deemed as an NVM subsystem. Each NVM subsystem may consist of one or more Endurance Groups (EG) to localize garbage collection. Each EG could contain a number of "NVM Sets" (NVMS), and each of those can consist of one or more Namespace (NS).

The *NVMe Storage Controller* (NSC) helps to connect a host with the NVM storage. An SSD can be accessed by a host locally as well as remotely. For remote access, a protocol called NVMe-oF (NVMe over fabric) has been defined that essentially extends the NVMe commands to be ported from host to the target and executed remotely (Minturn and Metz, 2015).

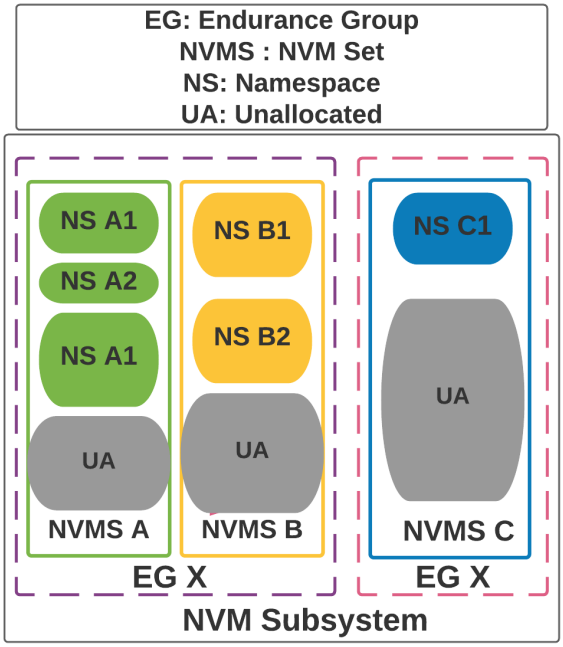


FIGURE 5.1: NVM Storage Hierarchy

The PLM is operated at the granularity of NVM-sets, which cycles between the aforementioned DTWin and NDWin as in Fig 5.2 such that all background activities are performed during the NDWin period. The **DTWin budget** can be represented by three attributes: (a) a predefined time limit, (b) a predefined limit on the number of reads, and (c) a predefined limit on the number of the writes that can be performed on the NVM-set. Both (b) and (c) are collectively termed as *Deterministic Counts (DC)* In this work. The NVM-set may transition to NDWin if either of these limits is exceeded. There are various DTWin attributes introduced in NVMe v1.4 such as DTWin read typical (C_{RD}), DTWin Write typical (C_{WR}), and DTWin time maximum (T_{DW}). We consider the parameters C_{RD} and C_{WR} as DTWin read deterministic count ($read\_DC$) and write deterministic count ($write\_DC$) respectively.

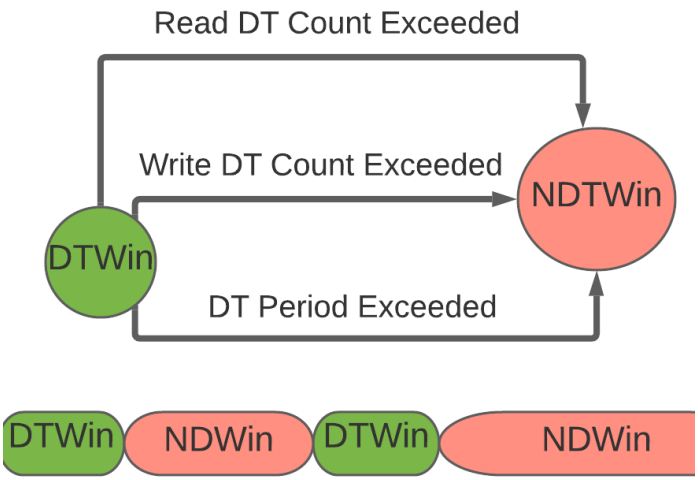


FIGURE 5.2: PLM Mechanism

5.2.2 PLM Feature And Its Functioning

The PLM feature proposed in (NVM, 2019) is primarily designed to cater to reads since the latencies for reads are generally more critical than for the writes. To achieve the deterministic latency, the host needs to obey the predictable latency operating rules. As stated earlier, to provide continuous access to DTWin mode to an application, we need multiple (two or more) NVM-sets that are identical and contain copies of the data. If all NVM-set copies are never simultaneously placed in the NDWin state, an application can read data in the DTWin mode any time. While the required duplication is expensive, it is needed only for the data accesses that must be deterministic; one copy suffices for all other data. Also, if multiple copies are maintained for resilience reasons, they can be used to provide the deterministic service as well. It is also worth noting that a traditional RAID1 (mirroring) does not provide the same functionality as PLM since the reads could occur in RAID1 while the SSD is doing the background operations.

The NVM-set may transition to the NDWin before exhausting its DTWin budget – this happens if there is an emergency management operation required such as an emergency garbage collection being triggered. The amount of time spent by the

NVM-set in the NDWin state depends on the necessary background operations for management activity (and the flushing of writes accumulated in the buffer.)

A remote host can setup a connection via NSC. The NSC advertises the DTWin related status for each NVM-set into a "log-page". The log-page is accessible to the host using NVMe commands and therefore the DTWin information per NVM-set. The host defines the beginning of DTWin and NDWin periods by making requests to the NSC using the "set-feature" command. Since a host could use multiple NVM-sets simultaneously, they could be at different points in their DTWin/NDWin cycles. Therefore, the host must influence DTWin/NDWin duration to synchronize multiple copies of NVM-sets. PLM's current specification puts the regular transition between DTWin and NDWin control with the host (if the log-page advertised status is respected) and the forced transition with the NSC (if the host does not respect the operating rules).

5.2.3 Coordinator Background

One of the initial issues we encountered in our study of the PLM feature was the associated problems while extending it to a multi-host environment. In such an environment a resource allocation module is required along with the existence of usual NSC capabilities. This module can be a part of the NSC itself; however, running separate workloads pertaining to the different hosts can create a bottleneck at the NSC. This overhead brings forth the need to break this module into sub-modules and place it outside the NSC in a suitable setting i.e. the storage server. This issue is explored in (Roy et al., 2021) by introducing the concept of a PLM Coordinator (PLMC), which acts like the aforementioned DTWin budget allocation entity that distributes $read\_DC/T_{DR}$. It interacts with the hosts and allocates for them their $read\_DC$ (depending on their QoS requirements) at the start of each DTWin period. The PLMC does not participate in the data path of the I/O requests, however, it is

also possible to place the PLMC on the host side as another host requesting for the shared NVMe Set.

5.2.4 Hybrid Storage Model

Hybrid storage model is quite popular this days for distinctive puposes. In (Hoseinzadeh, 2019) surveys with different set of storage technologies choices with their usefulness and pitfalls. The hybrid storage model with Optane memory, used in the same level with SSD or as a different level in the storage hierarchy, to enhance the performance and endurance of the overall storage system is explored by multiple researchers (Chen et al., 2020; Roy and Kant, 2020; Gunta et al., 2020). In (Chen et al., 2020), they have valuable findings to use for building emerging hybrid SSD controllers. They have observed, it is suitable to place small files to be placed in the Optane and large files on SSDs. The Optane is used to enhance the endurance of the overall storage system in (Roy and Kant, 2020). (Gunta et al., 2020) caches datablock based on the access eligibility of a block from SSD.

In our work, we have considered Optane in the same level as SSD to emulate the PLM feature. Therefore, there are no existing works with which we can compare the realized deterministic latency.

5.3 Achieving Deterministic Latency In Shared Environment

As stated earlier, PLMC is capable of distributing *read_DC* among the active hosts based on the application QoS. However, it doesn't consider write IOs. The shared environment becomes interesting when write intensive workloads are in play simultaneously and share the same working set. To assure consistency it is required to execute the write IO when requested. However, inexpert handling of write IO can consume the device *write_DC* and therefore can force DTWin to NDWin transition. This can result in a long spike in latency for latency sensitive applications. In this

work, we have proposed a coordinator - PLM^{light}Coordinator (PLMLC), to manage both read IO and the write IO in regular SSDs while embodying the features from PLM. Thus guaranteeing the low latency for high QoS applications.

5.3.1 Proposed PLMLC In A Shared Environment

We design the proposed PLM^{light}to distribute the *read_DC* among multiple hosts running different workloads that share the storage space while also handling write IOs. PLM^{light}has two sub-modules, the central coordinator agent i.e. PLMLC along with a *Write Handling Agent* (WHA) in each server. The PLMLC is placed on the control path, outside the storage controller. It acts as a host and communicates with the storage server using NVMe-OF. Therefore, it can collect the status of the PLM parameters from the NSC using NVMe commands. Also, it can collect the traffic experienced by each host via message passing to allocate *read_DC* for each host. The write IO needs to be executed immediately to maintain consistency and reliability. To control the *write_DC* usage of each NVM set, limited write IOs are performed on SSD. The excess write IOs need to be relocated in order to achieve deterministic latency. Therefore, the WHA needs to be placed in the storage server along with the NSC to direct the location of the write IO execution. In the proposed PLM^{light}, we need an additional device (which reports more deterministic write latencies) along with the regular SSDs to complete the aforementioned excess write IO, thus creating a *Hybrid Storage Model*.

In this work, we have considered Intel Optane SSD to accomplish these excess writes. The architecture of PLM^{light}is shown in the figure 5.3. PLMLC allocates *read_DC* to each host actively accessing the NVMe target, consisting of two or more NVM subsystem(s), i.e SSD(s), and Optane as the additional device. It also maintains a consolidated *write_DC* pertaining to the device so as to ensure the device's seamless transition to NDWin when *write_DC* is exhausted. Each subsystem can have one

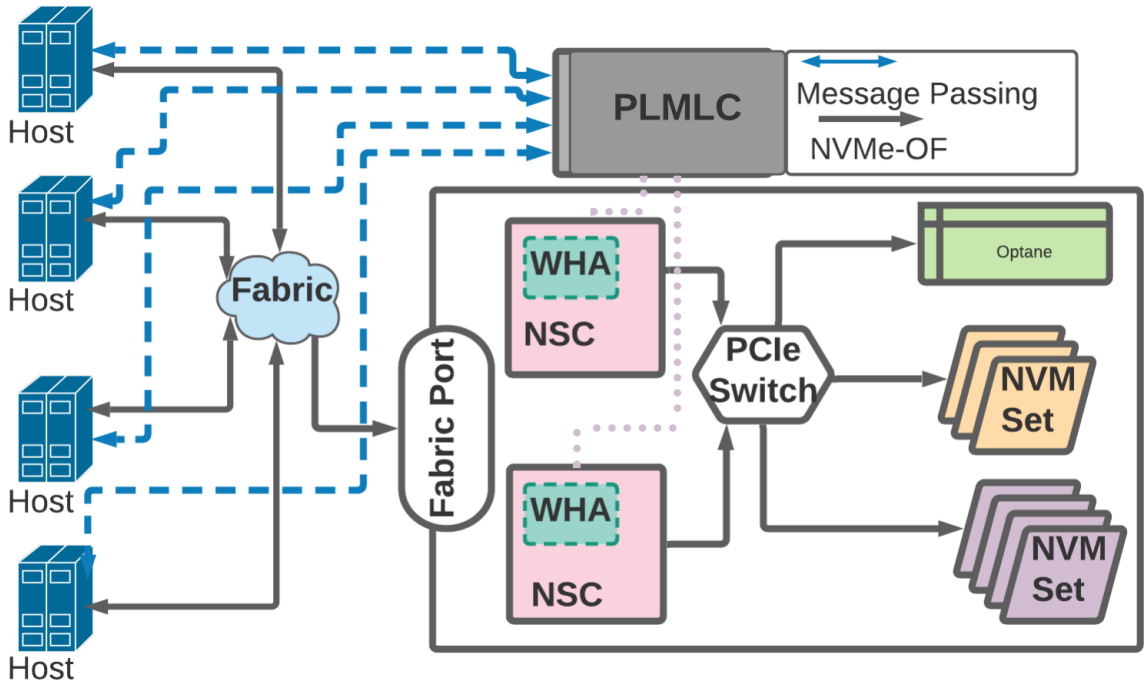


FIGURE 5.3: An Illustration of PLM^{light} Architecture

or more NSCs that control a number of NVM-sets. The hosts can communicate with PLMLC using conventional NVMe-OF protocol (Minturn and Metz, 2015; Guz et al., 2017) and access each NVM-set's log-page to collect read DC information. It is assumed that each host is reliable to convey its usage to PLMLC and respect the allocated limit. The PLMLC needs to learn the workload traffic characteristic to estimate the required DC. It allocates the *read_DC* to each host based on the *read_DC* estimation and the QoS class of the applications running on each hosts. Unlike PLMLC, there is no separate communication needed between hosts and WHA. The WHA decides the write IO execution device depending on the device characteristics.

In general, hosts with similar traffic characteristics will experience the same underlying device access latency (exclusive of any queuing or other latencies) in the absence of management operations. Therefore, the difference will be highlighted in tail latency. The end-to-end (E2E) latency will consist of different components, such as: (a) host-side IO dispatch and IO completion latency, (b) network transit latency,

(c) NVMe queuing and queue handling latency, and (d) device access latency. In reality, the QoS class definition would cover the end-to-end latency, and must be split into allowable latencies for each of these four components. For example, if TL is the overall desired 99% latency, we can split it into four pieces $TL_i, i = 1..4$ with $\sum_{i=1}^{4} = TL$. This paper is only focused on (d) following such a split.

5.3.2 Host Traffic Estimation By PLMLC

The PLMLC module of PLM$^{\text{light}}$needs the estimated *read_DC* of each host and distributes the available *read_DC* considering the application's QoS class. The estimation could be done either by each host individually and then communicated to PLMLC, or the PLMLC can do the estimation itself. The former is more cumbersome and perhaps not desirable since it adds extra burden on the hosts and could be subject to misbehavior if a host's estimation is faulty (e.g., a deliberate overestimation supplied by the application). However, the host can potentially use information that is not available to PLMLC, including perhaps the precise number of reads/writes to be performed during a DTWin.

The PLMLC can easily separate the requests from each host and do a statistical estimation of the traffic. However, the key difficulty in such an estimation is that the storage traffic is generally very bursty and nonstationary. Thus the standard quasi-stationary time-series prediction models, such as ARMA, Kalman-filters, etc. do not provide very good predictions (Weigend, 2018). In fact, our experiments with all these algorithms (not reported here) confirmed that these well-established methods did not provide any significant advantage over a simple exponential smoothing-based traffic estimation. In theory, more sophisticated neural net models could do better by capturing the patterns in the workload, however, this method has two key hurdles: (a) need for very long traces for training the model, and (b) need to retrain when workloads/applications change. Our results show that basic exponential smoothing

provides satisfactory results in spite of its simplicity.

Let $RR_j^{(m)}(n)$ denote the measured read request count (traffic) at the n-th DTWin for the j-th class and $RR_j^{(p)}(n)$ their smoothed estimated in the same period, with $0 < \zeta < 1$ as the smoothing constant. Then,

$$RR_j^{(p)}(n+1) = \zeta RR_j^{(m)}(n) + (1-\zeta)RR_j^{(p)}(n) \tag{5.1}$$

Where ζ is chosen experimentally as 0.5 for the entirety of each experiment.

5.3.3 Deterministic Count Allocations By PLMLC

For the DC allocation, we have closely followed our previous work in (Roy et al., 2021) where we considered k hosts running different workloads with different QoS classes represented as $Q_1, Q_2, ..., Q_k$. The n-th DTWin duration is termed $W_d(n)$ which is followed by the subsequent NDWin of duration $W_{nd}(n)$. The entirety of these two periods is collectively termed as the *scheduling period* $W(n)$, where

$$W(n) = W_d(n) + W_{nd}(n) \tag{5.2}$$

The write request behavior during a preceding $W_d(n)$ determines the duration of a succeeding $W_{nd}(n)$ as the writes may trigger background activities which in turn extend said $W_{nd}(n)$. During each $W(n)$ we need to allocate the total DC among the k hosts so as to establish differentiated treatment for all of $Q_1, Q_2, ..., Q_k$ classes and to also make sure that DC allocated is not wasted (i.e. a certain does not use its allocated DC)

5.3.4 Read DC Allocation

We term the total available $read\_DC$ during a $W_d(n)$ as C. If the predicted traffic results in a total required $read\_DC$ is below C, we then allocate the estimated value itself. However, if the predicted value exceeds C, we follow a fixed ratio style of allocation for the excess required $read\_DC$, which is as explained below.

Fixed Ratio Allocation Policy: In this allocation policy, the residual required *read_DC* is split among various hosts based on a predefined set of ratios r_j corresponding to the Q different classes. We further divide the allocation policy into two different styles so as to observe how it affects the QoS differentiation. This is done on the basis of how much minimum *read_DC* is guaranteed to each host. They are as follows -

- Equal Base (EB): In every DTWin, each QoS host starts with the same minimum *read_DC*. Though, additionally, they may get more *read_DC*s based on their traffic characteristics and QoS class.

- Prioritized Base (PB): At the beginning of each DTWin, each host is assigned *read_DC*s based on their QoS class. Additionally, more *read_DC*s are also allocated based on the QoS class, similar to *EM*.

In contrast to (Roy et al., 2021), we have considered only the fixed ratio policy as the other method mentioned in (Roy et al., 2021) (Strict Priority) and Fixed Ratio tend to perform similarly in a variety of situations.

5.3.5 Write DC Management

We can distribute the *read_DC* considering the traffic intensity and QoS class of the workload. In that case, we can force low QoS class workloads to wait and perform their execution during NDWin, when allocated read DC is exhausted before the DTWin ends. In the case of *write_DC*, we cannot stall the write operations until DTWin ends as the working set is shared among all the hosts. This action will violate consistency. Therefore writes need to be performed immediately. This is also the reason why we refrained from differentiating between QoS classes for allocating *write_DC*. To mitigate the premature end of DTWin, we will reduce the number of writes going to the SSD. Here we consider a policy where small writes ($4K$ to $64K$ IO size in our

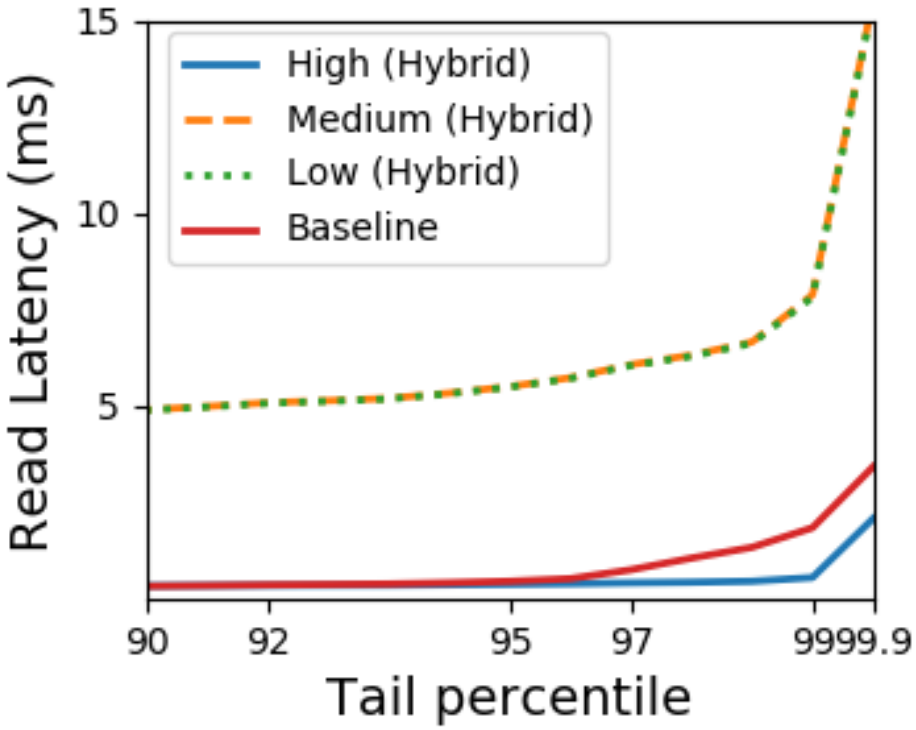


FIGURE 5.4: Read Tail Latency Comparison (W-II)

case) are done in Optane and large writes (larger than $64K$) in SSD.

5.4 Evaluation Of The Proposed PLMLC

5.4.1 Workloads And Configurations Used

To evaluate our mechanism, we use the widely accepted FIO benchmarking tool to create workloads that can stress test PLMLC (Axboe, 2013). We ran FIO with four different configurations which we denoted as I-IV. These workloads were spawned into three different processes with each process pinned to a different CPU. The CPUs are considered as three hosts in our description. The workload configurations are as follows.

1. **Workload I and II:** The first two workloads have a read-write ratio of 30:70 and 70:30 respectively. Both of these workloads have a request size of 4KB. This tests PLMLC 's performance for single block requests.

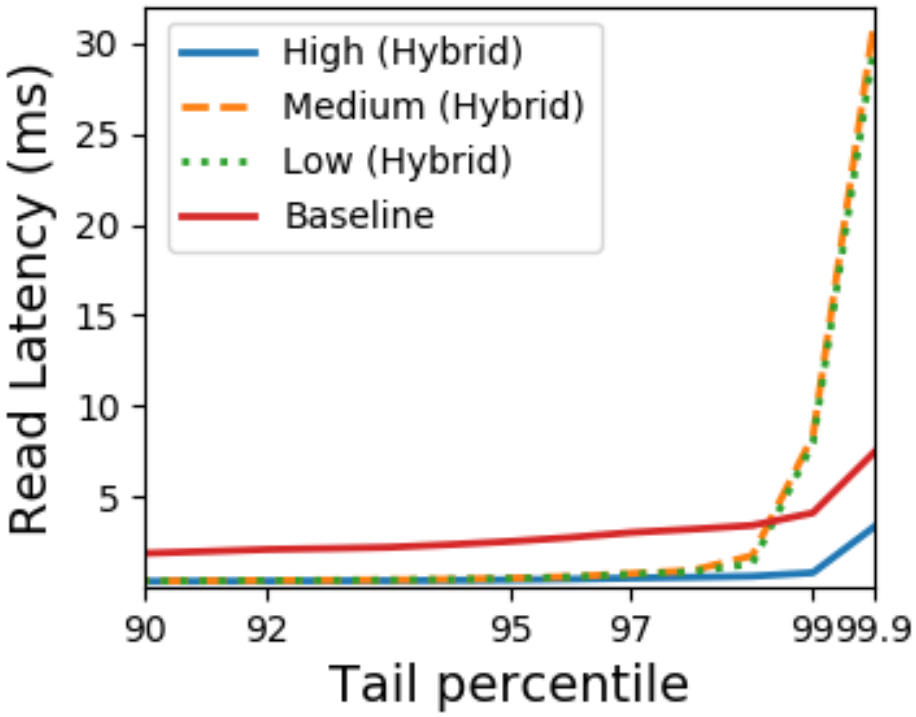


FIGURE 5.5: Read Tail Latency Comparison (W-IV)

Workload	High	Medium	Low
$WorkloadI$	421.6	424.7	426.3
$WorkloadII$	383	397.2	396
$WorkloadIII$	1161.6	1163.2	1160.4
$WorkloadIV$	1117.3	1131.1	1132.6

Table 5.1: Average IO in KB/DTWin

2. **Workload III and IV:** Here, the request size for these two workloads range from 4KB to 128KB, but the same 30:70 and 70:30 read/write ratios are maintained. This tests PLMLC's performance for variable and larger request sizes.

5.4.2 Workloads And Configurations Used

Due to pervasive use of caching and prefetching throughout the storage hierarchy, we considered primarily 100% random workloads. This presents the most challenging workload for PLMLC. We can observe this random behavior in Figs 5.11 and 5.12 and can also observe how the variable request size of Workload IV ensures that the

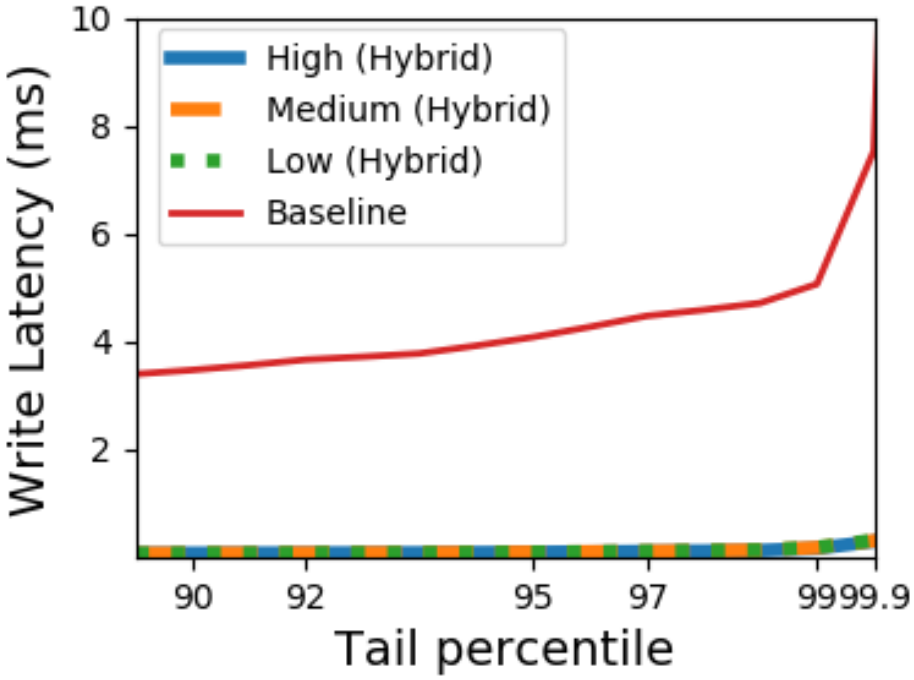


FIGURE 5.6: Write Tail Latency Comparison (W-IV

rate of requests is smaller as compared to that for Workload I. Both these figures show a two-second snapshot of workloads I and IV.

We have used the Linux applications *blktrace* and *blkparse* to capture the replayable trace pertaining to each of the workloads generated by FIO. The resultant trace contained four actions (in the given order) pertaining to each request -

1. **Q / Queue Request:** This action notes the intent to queue the request at the given location. No request actually exists yet.

2. **G / Get Request:** This action corresponds to allocating a *struct request* container, which is a mandatory step to send any type of request to the block device.

3. **D / Issue Request:** As the name suggests, this action issues the created existing request to the driver.

4. **C / Completed Request:** Finally, this notifies the completion of the request.

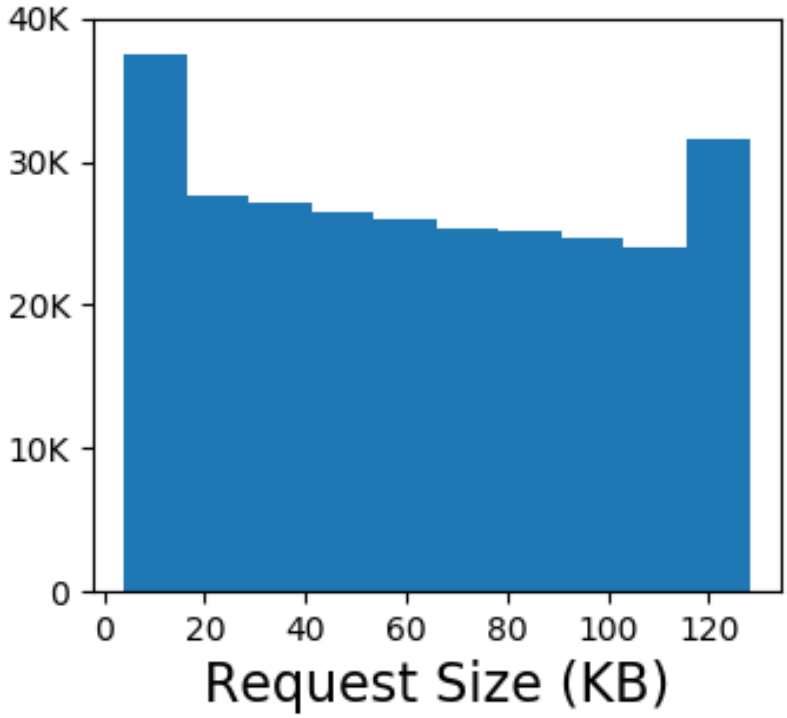


FIGURE 5.7: Histogram depicting request size distribution in W-III

For our evaluation, we have only considered the action **D** as that is the only action of significance to us. This is because every other action is dependant on how the application itself behaves, for example, the request may be completed at a different time in FIO compared to PLMLC. The final trimmed traces (with a single action pertaining to a single request) were used for our experimentation.

5.4.3 Evaluation Metrics

In this work, our goal is to achieve low latency with minimum variation. Therefore, measuring and comparing tail latencies, along with the average latencies, are useful here.

- Read Tail Latency: For each of the considered workloads, read latencies are measured and tail latency at different %-tile, viz. 90%-tile, 95%-tile, 99%-tile, 99.9%, are reported. These values represents the maximum latency value, for the fastest 90%, 95%, 99% and 99.9% of read IO requests respectively.

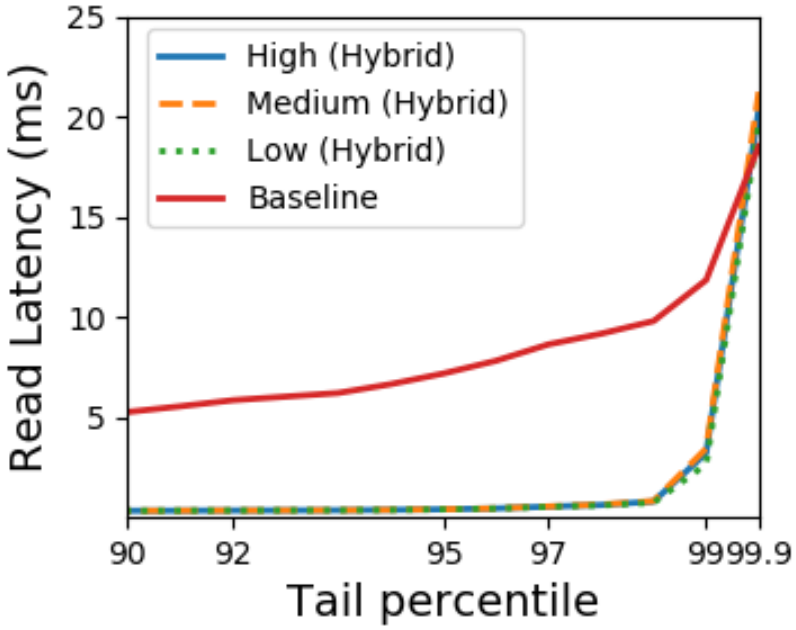


FIGURE 5.8: Read Tail Latency Comparison (EB)

- Write Tail Latency: Similar to the read latency, write latencies are measured and tail latencies are reported.

5.4.4 Experimental Setup

In this work, to achieve our goal of deterministic latency, we have emulated the NVMe 1.4 proposed PLM using workloads containing both reads and writes in a distributed shared environment. We have considered two identical SSDs, NVMe SSD 970 EVO Plus, as NVM-sets. The Optane memory on PCIe is considered as the write buffer to these NVM sets. As there is no common driver application to manage both SSDs and Optane memory, we have implemented an application that manages both SSDs and Optane in a Linux environment, Ubuntu 20.04. We have turned off the SSDs internal write cache and Linux page cache to reduce the effect of caching mechanism in the experienced latency of an IO request.

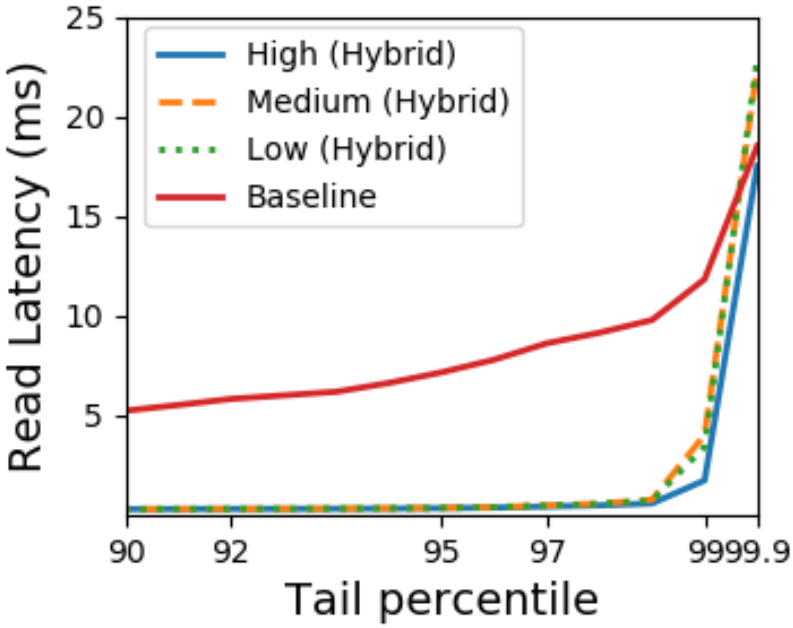


FIGURE 5.9: Read Tail Latency Comparison (PB)

5.4.5 Evaluation Results

For evaluation, we have considered the above four workloads(I-IV). Each of these workload types runs on three hosts with different QoS class requirements. We have performed a baseline experiment with three hosts running simultaneously for each workload on SSD without any QoS-based treatment. As there is no differential treatment for the three hosts in the Baseline scenario, all three exhibit close to identical behavior with negligible difference. Hence we have only depicted one of the hosts in our figures as the *Baseline* comparison. We compared our PLM^{light}performance for all the workloads against the baseline. We performed the PLM^{light}performance tests using the fixed ratio policy with EB and PB. For each of the workloads, we considered three hosts in the *Hybrid Storage Model* (as mentioned in Section IIIA) with different QoS classes (high, medium and low) running together.

For each of the workloads, we considered consistent configurations depending on the workload characteristics. The available *read_DC* and *write_DC* for every DTWin are the same and considered as 80% of the average *read* and *write* IOs

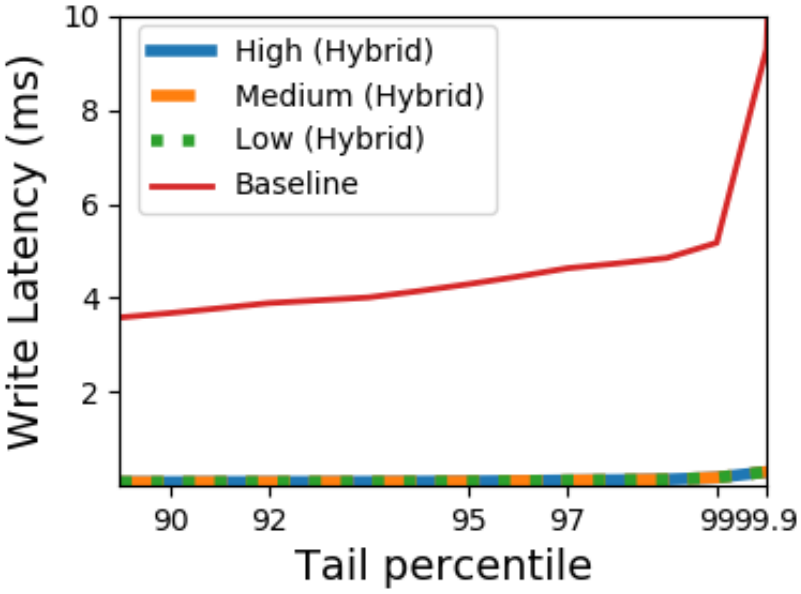


FIGURE 5.10: Write Tail Latency Comparison

per DTWin. The DTwin period is considered as 1 millisecond. Therefore, 80% of the *read* and *write* IOs will be served in 1 millisecond. We want to examine how efficiently PLM$^{\text{light}}$can distribute this DTWin IO capacity among three classes. The PLM$^{\text{light}}$performance tests are done considering both EB and PB, where total base

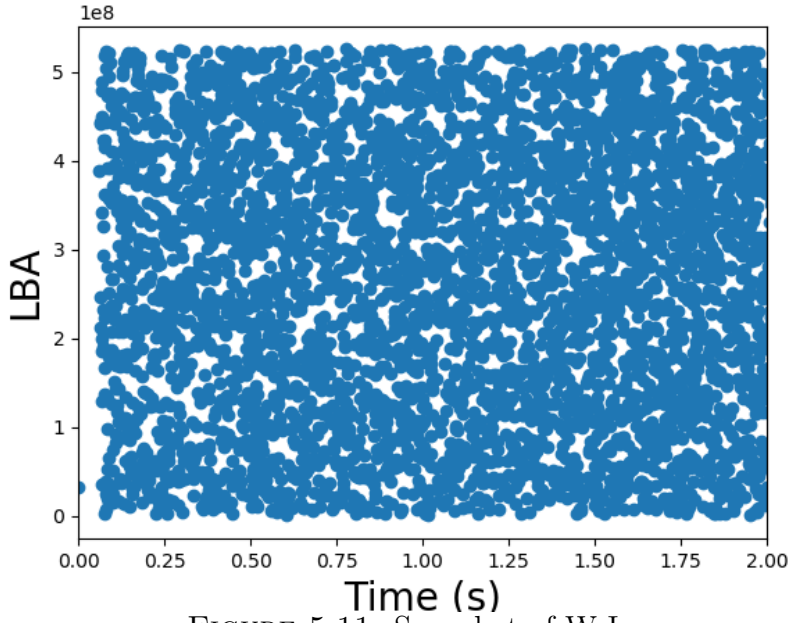


FIGURE 5.11: Snapshot of W-I

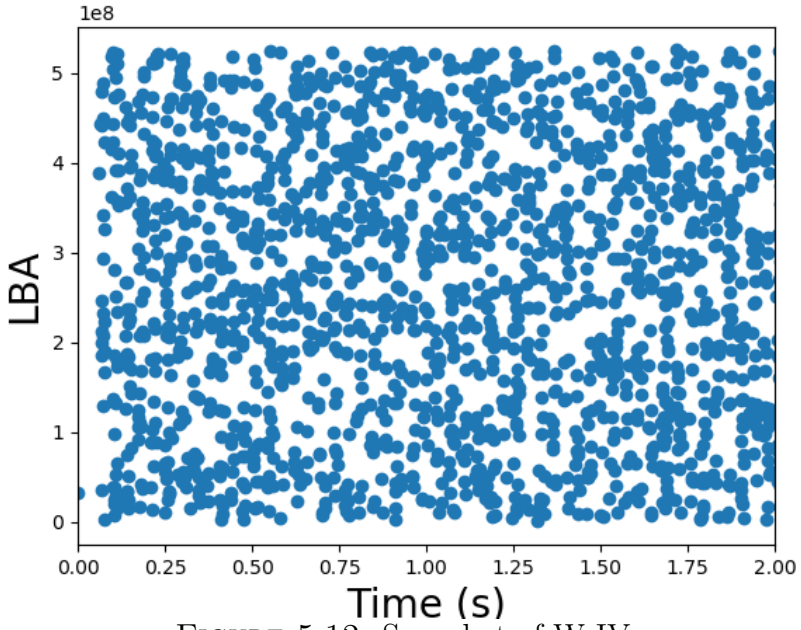


FIGURE 5.12: Snapshot of W-IV

read_DC collectively assigned for high, medium and low are 75% of the average *read*. In the case of EB, the total base *read_DC* is shared among three QoS equally as mentioned in Section 5.3.3. In contrast, total base *read_DC* is shared between QoS classes as 50%, 30% and 20% for high, medium and low respectively. Therefore, for a high IO intensive application, all QoS class have the possibility to achieve the required amount of DC to experience low read latency. PLM^{light}doesn't distribute *write_DC* among the QoS classes. All QoS classes share the *write_DC* on a first-come-first-serve basis.

5.4.6 Write-intensive IO Workload

Write intensive workloads (Workloads I and III) running on SSD can yield a few milliseconds read latency as shown in Figs 5.8 and 5.9 and considered as baseline. Therefore, it is reasonable to compare our PLM^{light}considering the *Hybrid Storage Model* with the baseline. Baseline has millisecond-level tail latency from 90 percentile and beyond. Whereas, PLM^{light}has microsecond-level tail latency at 99%-tile. The

tail touches the millisecond-level latency at 99.9%-tile, which is not different than baseline. At 99%-tile, the High QoS host under PLM^{light}in Workload III improves 2.7x and 5.8x than baseline for EM and PM respectively. With these workload characteristics and considered DC configuration, there is not much differentiated service provided with EM policy. However, for PM policy, PLM^{light}provides significant differentiated service. Therefore, the High QoS host performs better by 0.5x compared to the Low and Medium QoS host for 99%-tile tail latency. The write latency also significantly improves with PLM^{light}as shown in fig 5.10. It happens as Optane consumes a large portion of writes. The IO size distribution of this workload is represented in fig 5.7. All writes with IO size greater than 64K are consumed by the SSD. Otherwise, it is persisted in the Optane. We cannot see differentiated service for write latency among the three QoS classes as $write_{DC}$'s are used collectively.

Further, we wanted to explore the write intensive workload performance with PLM^{light}, where all writes are small (4KB) i.e. Workload I. Fig 5.13 shows that PLM^{light}can outperform baseline for High QoS. It improves close to 0.5x at 99.9%-tile. However, the differentiated service for high QoS results in higher latency for low and medium QoS in this Workload. Moreover, when we consider the writes, the latency for all the QoS classes stays at microsecond values as shown in fig 5.14. This happens as writes are all consumed by the Optane buffer and transferred to SSD during the NDWin period.

5.4.7 Read-intensive IO Workload

Although PLM^{light}is primarily focused on write intensive workloads, we wanted to conduct performance tests considering the read intensive workloads (Workloads II and IV) too. Fig 5.4 shows that high QoS read tail latency performs better than baseline by 70% at 99%-tile and 30% at 99.99%-tile with a hybrid model. The primary reason of this small improvement in read latency is the small portion of write data

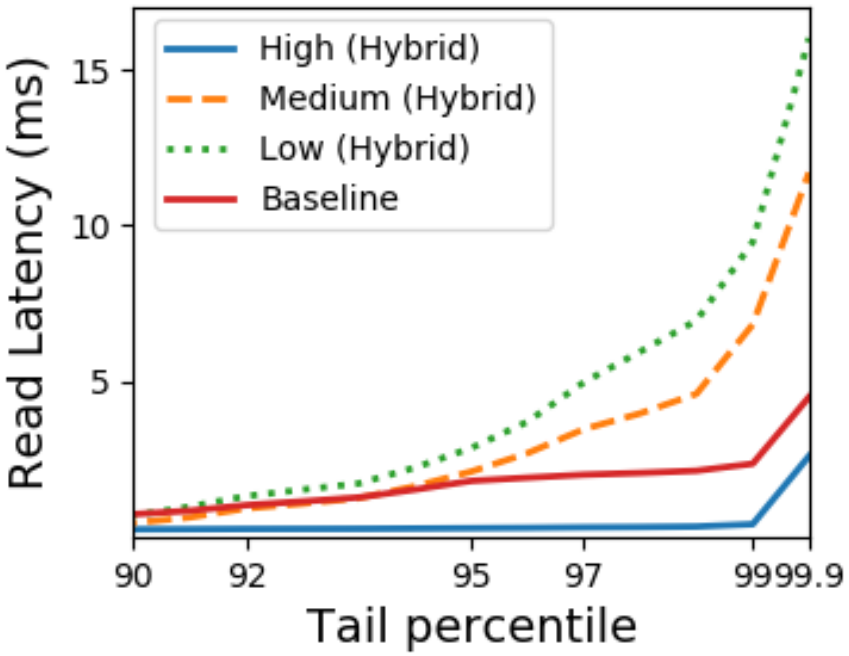


FIGURE 5.13: Read Tail Latency Comparison for W-I

goes to Optane and results in a spike in SSD latency. Therefore, any amount of writes dissociated from SSD can yield better read tail latency. This finding can be made stronger with a performance test with significant IO size variations even in read intensive workload.

In this workload (Workload IV), we have variations in IO size, and a small fraction of writes with less than 64KB IOSize persist in the Optane, and the rest go to the SSD. In fig 5.5, the high QoS outperforms the baseline read tail latency by 80% for 99%-tile and 55% for 99.9%-tile, which is almost similar to the performance achieved with the read-intensive small IO workload (Workload II). It is noticeable that low and medium QoS class read latency improves from the other read-intensive workload (Workload II) by avoiding write IO execution in SSD. Although, a small amount of write is present in the workload, persisting a portion of it in Optane improves the write tail latency significantly in PLM^{light}as compared to the baseline.

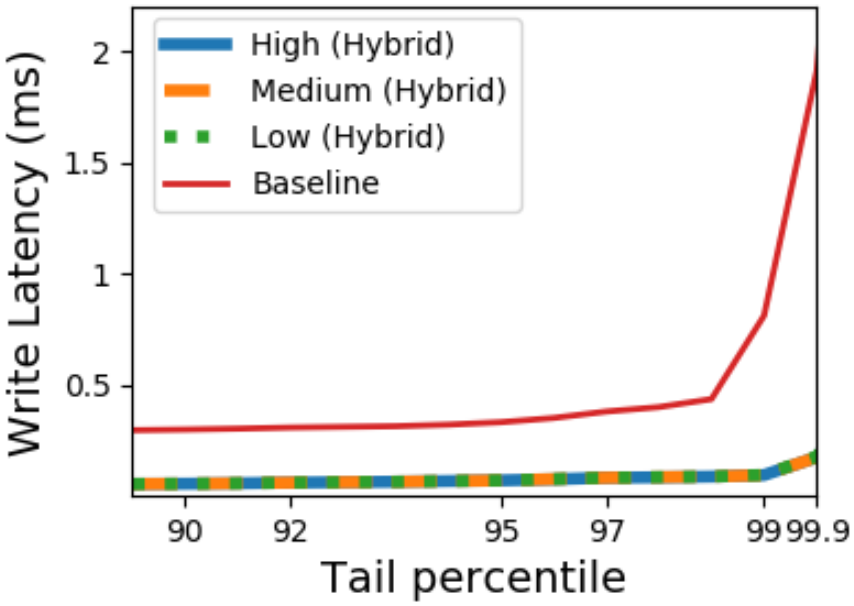


FIGURE 5.14: Write Tail Latency Comparison for W-I

5.5 Conclusions and Future Work

In this work we have considered a mechanism to enhance the determinism of SSD access latency by making use of DTWin/NDWin concepts proposed in NVMe1.4, coupled with the introduction of a more deterministic device in the overall architecture. Since no commercially available SSDs currently have yet implemented this feature, and there is a huge installed base of SSDs, our proposed mechanism instead attempts to increase determinism by artificially defining DTWin/NDWin periods such that minimal writes to SSDs are issued during the DTWin period. Instead, most writes are directed to an Intel Optane device that inherently provides deterministic latency. These writes are opportunistically transferred to SSD during NDWin periods. In this work, we design a coordinator that can handle requests from multiple hosts to a shared device. The coordinator distributes the allowed IO counts to individual hosts based on their QoS requirements. A detailed evaluation shows that mechanism can substantially enhance the determinism and also reduce the write access latencies by close to 5.8x.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this thesis, we have addressed the high-performance requirements of emerging data-intensive applications. Multi-bit/cell-based SSDs are improving and replacing traditional HDDs in the storage hierarchy to achieve low latency. However, the enormous amount of write IOs can create latency peaks touching a few tens of milliseconds and immense stress on SSD's endurance. The emerging new technology in storage and storage access protocols is considered to control the latency peaks tightly. Designing hybrid storage, including SSD and ENVM, can lower the stress on SSD. Moreover, the hybrid storage model with a suitable mechanism can diminish the unusual latency peaks for latency-critical applications.

Maintaining the endurance rating of a NAND device is crucial, and therefore, the amount of write IO completion needs to be handled cautiously. In our first research contribution, we have addressed the opportunities and challenges in using emerging NVM technologies to manage the trade-offs between performance and endurance characteristics of low endurance SSD-based high-performance storage. The availability of high-speed technologies like Intel Optane also allows immediate persist of modified data and thereby eliminates (or at least substantially reduces) data loss

potential and simplifies resilience. We discussed several ways of combining NAND flash technologies and ENVM technologies. The immediate persist reduces the read latency, which is often more critical than the write latency. Of course, the overall write latency increases (since all writes must eventually go to the SSD), but this is not important since a write completion response can be returned to the application as soon as the ENVM persist is done. The high-performance computing applications often involve large writes, and the actual price of immediate persist in potentially degrading the endurance of the ENVM. However, since EVNM has a very high PE cycle rating, a larger size can often address the problem. In the future, we plan to explore other policies for storage hierarchies involving SSD and ENVM, including byte-addressable ones, and will also explore performance for a more extensive set of workloads.

The next goal was to achieve deterministic latency in read-oriented workload. We have explored and extended the predictable latency mode (PLM) feature introduced in NVMe v1.4. We show how the PLM concept can be extended to a multi-host environment where multiple hosts share an NVMe device, as would be the case for database accesses. For this, we have proposed a PLM coordinator (PLMC) that runs on a storage server and can be implemented without any changes to the current NVMe v1.4 standard. We evaluated the proposed PLMC with extensive experiments performed in the SSD simulator (MQSim). The experimental evaluation of the mechanism demonstrates that it can substantially reduce tail latency and help deliver predictable latency to the higher QoS classes with 93.2% improvement in 99.99 percentile tail latency compared to the shared environment. In the future, we will also examine the impact of writes regarding their effect on the duration of the non-deterministic window and their impact on the deterministic mode operation by filling up the NVRAM buffer and forcing a switch-over to the non-deterministic mode.

The subsequent interest was to guarantee deterministic latency when write-

intensive workloads are running together with read-intensive workloads. We have considered a mechanism to enhance the determinism of SSD access latency by using DTWin/NDWin concepts proposed in NVMe1.4, coupled with the introduction of a more deterministic device in the overall architecture. Since no commercially available SSDs have yet implemented this feature, and there is a substantial installed base of SSDs, our proposed mechanism instead attempts to increase determinism by artificially defining DTWin/NDWin periods such that minimal writes to SSDs are issued during the DTWin period. Instead, most writes are directed to an Intel Optane device that inherently provides deterministic latency. These writes are opportunistically transferred to SSD during NDWin periods. This contribution designs a coordinator that can handle requests from multiple hosts to a shared device. The coordinator distributes the allowed IO counts to individual hosts based on their QoS requirements. A detailed evaluation shows that mechanism can substantially enhance the determinism and reduce the write access latencies by close to 5.8x.

6.0.1 Improve PLM$^{\text{light}}$

We have observed it is needed to use access latency as feedback to track the actual SSD background operation. Tracking the request latency can be useful to switch to the SSD that is not performing any background operation currently. We can use a latency monitor module (LM), as shown in fig 6.1, to track the access latency of each request for any SSD. The request experiencing the highest latency can help us to infer the actual background operations when it exceeds a threshold latency. This insight will be useful to improve the low latency better than the current PLM$^{\text{light}}$.

6.0.2 Other Future Opportunities

Storage systems form a crucial component of data center infrastructure and substantially influence the performance of the applications. With applications

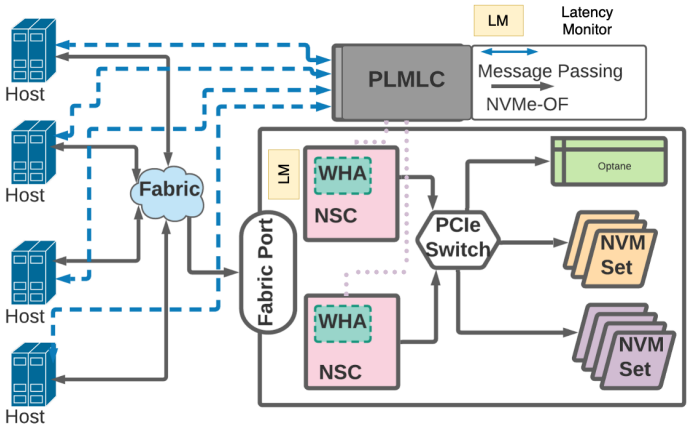


FIGURE 6.1: PLM^{light}with Latency Module

becoming ever more data-centric, this reliance on a fast storage system that can provide the required quality of service (QoS) treatment becomes paramount.

Despite the apparent simplicity, large-scale storage systems are incredibly complex beasts with little visibility for application into storage side complexities and little visibility for the storage system into the application side. Furthermore, nearly all storage resides away from the hosts in "storage servers", which means that network is an indispensable part of any storage access. This state of affairs, coupled with rapid advancement in storage technologies, makes the issue of end-to-end (E2E) QoS for storage systems crucial and challenging. The round-trip of a request-response pair can be considered as E2E QoS.

Fig. 6.2 shows the path of a storage request. The IO request is initiated from a host and travels to the target via the network. The response generated comes back to the host as shown in 6.2 . As stated earlier, a consistent QoS treatment is essential through all the components to avoid anomalies. For example, a QoS mechanism that gives preference to the small size transfers through the network may transfer all read requests to the target quickly (regardless of the write traffic going in the direction of read requests), process them at the target according to their priority, but fail to get the high priority data back to the host quickly through the network.

The opportunities for future research are as follows:

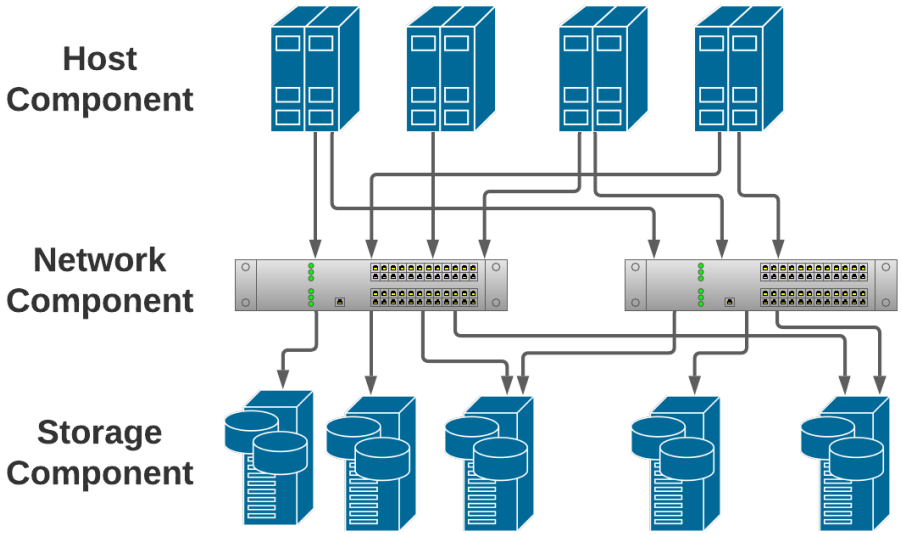


FIGURE 6.2: E2E Component

- End-to-End QoS Consideration: The round trip of request-response pair is one way to think about E2E QoS. It can be considered as a point-to-point view of QoS. Other scenarios also exist but become increasingly more complex to handle. For example, an application running on a host may need data from multiple targets that potentially reside in different storage servers (a point-to-multipoint scenario). Thus if we consider QoS from the entire application's perspective, all such paths from the host to different targets require consistent treatment. For example, if the data comes both from a slow and fast target (e.g., HDD vs SSD), it may suffice (and even desirable) to put the fast target request into a lower QoS class. If such assignment is done statically, no explicit coordination is needed; however, if the class assignment is done dynamically to cope with a varying load on the device or the network, QoS management becomes crucial. We may need to do this dynamically depending on the loading levels of the target and the network path, which requires a level of coordination.

- End-to-End Hinting: We need to define a rather broad class of QoS classes from the perspective of application requirements. A class could be identified in

many ways. One possibility, closest to traditional QoS specification, is to specify what type of service we want for the various QoS classes. A significant issue entangled with End-to End is the need for hints originating at different points and consumed at others. In particular, not every hint goes from the application (or host OS) to the end device. It needs attention and decision from any point in the path.

- Binding Storage and Network solutions together at different components: There is an ample number of proposed QoS solutions by multiple researchers. Also, storage industry contributions can be classified in multiple areas, such as SSD technology, persistent memory with SSD, NVMe storage protocols, smart NICs, and many more. On the network front, we have several proposed differentiated services for various QoS classes. Stitching different solutions together at multiple points is essential for a given application requirement.

We hope that this dissertation and the challenges highlighted here will open new avenues for future research ideas.

BIBLIOGRAPHY

(2019), "NVM Express Base Specification, Rev 1.4," https://nvmexpress.org/wp-content/uploads/NVM-Express-1_4-2019.06.10-Ratified.pdf.

(2020)https://www.intel.com/content/dam/www/public/us/en/documents /technology-briefs/low-latency-for-storage-intensive-workloads-tech-brief.pdf.

Abraham, M. et al. (2010), "NAND flash trends for SSD/Enterprise," *Flash Memory Summit*.

Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J. D., Manasse, M. S., and Panigrahy, R. (2008), "Design tradeoffs for SSD performance." in *USENIX Annual Technical Conference*, vol. 57, Boston, USA.

Alsalibi, A. I., Mittal, S., Al-Betar, M. A., and Sumari, P. B. (2018), "A survey of techniques for architecting SLC/MLC/TLC hybrid Flash memory–based SSDs," *Concurrency and Computation: Practice and Experience*, 30, e4420.

Anderson, P., Black, R., Cerkauskaite, A., Chatzieleftheriou, A., Clegg, J., Dainty, C., Diaconu, R., Drevinskas, R., Donnelly, A., Gaunt, A. L., et al. (2018), "Glass: A new media for a new era?" in *USENIX HotStorage*.

Axboe, J. (2013), "Fio," .

Bednar, P. and Katos, V. (2011), "SSD: New challenges for digital forensics," in *Proceedings of the 8th Conference of the Italian Chapter of the Association for Information SystemsItAIS*.

Bornholt, J., Lopez, R., Carmean, D. M., Ceze, L., Seelig, G., and Strauss, K. (2016), "A DNA-based archival storage system," *ACM SIGARCH Computer Architecture News*, 44, 637–649.

Cafferty, B. J., Ten, A. S., Fink, M. J., Morey, S., Preston, D. J., Mrksich, M., and Whitesides, G. M. (????), "Storage of Information Using Small Organic Molecules," *ACS central science*.

Cai, Y., Ghose, S., Haratsch, E. F., Luo, Y., and Mutlu, O. (2017), "Error characterization, mitigation, and recovery in flash-memory-based solid-state drives," *Proceedings of the IEEE*, 105, 1666–1704.

Chang, L.-P., Kuo, T.-W., and Lo, S.-W. (2004), "Real-time garbage collection for flash-memory storage systems of real-time embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, 3, 837–863.

Chen, F., Koufaty, D. A., and Zhang, X. (2011), "Hystor: making the best use of solid state drives in high performance storage systems," in *Proceedings of the international conference on Supercomputing*, pp. 22–32.

Chen, H., Lv, Y., Li, C., Gu, S., and Shi, L. (2020), "An Empirical Study of Hybrid SSD with Optane and QLC Flash," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pp. 175–178, IEEE.

Chen, S., Delimitrou, C., and Martínez, J. F. (2019), "Parties: Qos-aware resource partitioning for multiple interactive services," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 107–120.

Cobb, D. and Huffman, A. (2012), "Nvm express and the pci express ssd revolution," .

Dean, J. and Barroso, L. A. (2013), "The tail at scale," *Communications of the ACM*, 56, 74–80.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. (2007), "Dynamo: Amazon's highly available key-value store," *ACM SIGOPS operating systems review*, 41, 205–220.

Do, J., Lomet, D., and Picoli, I. L. (2019), "Improving CPU I/O performance via SSD controller FTL support for batched writes," in *Proceedings of the 15th International Workshop on Data Management on New Hardware*, pp. 1–8.

Du, Y., Jia, Y., Zhang, M., Zeng, J., and Xue, C. J. (2017), "Enhancing SSD performance with LDPC-aware garbage collection," in *2017 IEEE 6th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, pp. 1–4, IEEE.

Eshghi, K. and Micheloni, R. (2013), "SSD architecture and PCI Express interface," in *Inside solid state drives (SSDs)*, pp. 19–45, Springer.

Fan, Z., Wu, F., Park, D., Diehl, J., Voigt, D., and Du, D. H. (2017), "Hibachi: A cooperative hybrid cache with nvram and dram for storage arrays," in *Proc. 33rd Int. Conf. Massive Storage Syst. Technol.(MSST)*.

Feldman, T. and Gibson, G. (2013), "Shingled magnetic recording: Areal density increase requires new data management," *USENIX; login: Magazine*, 38, 22–30.

Gal, E. and Toledo, S. (2005), "Algorithms and data structures for flash memories," *ACM Computing Surveys (CSUR)*, 37, 138–163.

García, A. A., de Jong, R., Wang, W., and Diestelhorst, S. (2017), "Composing lifetime enhancing techniques for non-volatile main memories," in *Proceedings of the International Symposium on Memory Systems*, pp. 363–373, ACM.

Gharehchopogh, F. S. and Arjang, H. (2014), "A survey and taxonomy of leader election algorithms in distributed systems," *Indian Journal of Science and Technology*, 7, 815.

Gill, B. S. and Modha, D. S. (2005), "SARC: Sequential Prefetching in Adaptive Replacement Cache." in *USENIX Annual Technical Conference, General Track*, pp. 293–308, Washington, D.C., USENIX.

Gunnam, K. K. (2017), "Wear leveling in non-volatile memories," US Patent App. 15/627,135.

Gunta, J., Kant, K., and Abouelwafa, A. (2020), "FussyCache: A Caching Mechanism for Emerging Storage Hierarchies," in *2020 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 74–81, IEEE.

Guo, J., Hu, Y., Mao, B., and Wu, S. (2017), "Parallelism and garbage collection aware I/O scheduler with improved SSD performance," in *2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 1184–1193, IEEE.

Guz, Z., Li, H. H., Shayesteh, A., and Balakrishnan, V. (2017), "NVMe-over-fabrics Performance Characterization and the Path to Low-overhead Flash Disaggregation," in *ACM SYSTOR*, pp. 16:1–16:9.

Guz, Z., Li, H., Shayesteh, A., and Balakrishnan, V. (2018), "Performance characterization of nvme-over-fabrics storage disaggregation," *ACM Transactions on Storage (TOS)*, 14, 1–18.

Hoseinzadeh, M. (2019), "A survey on tiering and caching in high-performance storage systems," *arXiv preprint arXiv:1904.11560*.

Hu, Y., Jiang, H., Feng, D., Tian, L., Luo, H., and Zhang, S. (2011), "Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity," in *Proceedings of the international conference on Supercomputing*, pp. 96–107.

Huang, S.-M. and Chang, L.-P. (2018), "Providing SLO compliance on NVMe SSDs through parallelism reservation," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 23, 1–26.

Huffman, A. (2012), "Nvm express," *revision 1.0 c. Intel Corporation*.

Hwang, E., Park, J., Rauschmayer, R., and Wilson, B. (2017), "Interlaced magnetic recording," *IEEE Transactions on Magnetics*, 53, 1–7.

Intel (2015), "Intel and Micron produce breakthrough memory technology," Intel Newsroom.

Intel (2018), "Intel optane Technology," Retrieved November, 02, 2018.

J. Gim, Y. Won, T. H. and Kant, K. (2015), "SmartCon: Smart Context Switching for Fast Block Devices," *ACM transactions on Storage, Vol 11, Issue 2*.

Kim, J., Lim, K., Jung, Y., Lee, S., Min, C., and Noh, S. H. (2019), "Alleviating garbage collection interference through spatial separation in all flash arrays," in *USENIX ATC*, pp. 799–812.

Kim, S. and Kwak, J. W. (2019), "Analysis of Potential Risks for Garbage Collection and Wear Leveling Interference in FTL-based NAND Flash Memory," *Journal of The Korea Society of Computer and Information*, 24, 1–9.

Koo, D., Kim, J.-S., Hwang, S., Eom, H., and Lee, J. (2017), "Adaptive hybrid storage systems leveraging SSDs and HDDs in HPC cloud environments," *Cluster Computing*, 20, 2119–2131.

Lee, C., Kumano, T., Matsuki, T., Endo, H., Fukumoto, N., and Sugawara, M. (2017), "Understanding Storage Traffic Characteristics on Enterprise Virtual Desktop Infrastructure," in *Proceedings of the 10th ACM International Systems and Storage Conference*, SYSTOR '17, pp. 13:1–13:11, New York, NY, USA, ACM.

Lin, M., Chen, R., Lin, L., Li, X., and Huang, J. (2018), "Buffer-Aware Data Migration Scheme for Hybrid Storage Systems," *IEEE Access*, 6, 47646–47656.

Liu, D., Zhong, K., Wang, T., Wang, Y., Shao, Z., Sha, E. H.-M., and Xue, J. (2016a), "Durable address translation in PCM-based flash storage systems," *IEEE Transactions on Parallel and Distributed Systems*, 28, 475–490.

Liu, N., Cope, J., Carns, P., Carothers, C., Ross, R., Grider, G., Crume, A., and Maltzahn, C. (2012), "On the role of burst buffers in leadership-class storage systems," in *012 ieee 28th symposium on mass storage systems and technologies (msst)*, pp. 1–11, IEEE.

Liu, Y., Gunasekaran, R., Ma, X., and Vazhkudai, S. S. (2016b), "Server-side log data analytics for I/O workload characterization and coordination on large shared storage systems," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 819–829, IEEE.

Lo, J., Phan, L. D., and Pajaro, C. (2014), "Managing wear leveling and garbage collection operations in a solid-state memory using linked lists," US Patent 8,713,066.

Lu, Y., Shu, J., Sun, L., and Mutlu, O. (2017), "Improving performance and endurance of persistent memory with loose-ordering consistency," *IEEE Transactions on Parallel and Distributed Systems*.

Luu, H., Winslett, M., Gropp, W., Ross, R., Carns, P., Harms, K., Prabhat, M., Byna, S., and Yao, Y. (2015), "A multiplatform study of I/O behavior on petascale supercomputers," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, pp. 33–44.

Ma, D., Feng, J., and Li, G. (2014), "A survey of address translation technologies for flash memories," *ACM Computing Surveys (CSUR)*, 46, 1–39.

Megiddo, N. and Modha, D. S. (2003), "ARC: A Self-Tuning, Low Overhead Replacement Cache." in *FAST*, vol. 3, pp. 115–130, Washington, D.C., USENIX.

Micheloni, R. and Crippa, L. (2017), "Solid State Drives (SSDs)," in *Solid-State-Drives (SSDs) Modeling*, pp. 1–17, Springer.

Micheloni, R., Marelli, A., and Eshghi, K. (2013), *Inside solid state drives (SSDs)*, Springer.

Minturn, D. and Metz, J. (2015), "Under the hood with NVMe over Fabrics," in *Ethernet Storage Forum. SNIA*.

Mittal, S. and Vetter, J. S. (2016), "A Survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems," *IEEE Transactions on Parallel and Distributed Systems*, 27, 1537–1550.

Mohan, V., Siddiqua, T., Gurumurthi, S., and Stan, M. R. (2010), "How I Learned to Stop Worrying and Love Flash Endurance." *HotStorage*, 10, 3–3.

Neal, I., Reeves, B., Stoler, B., Quinn, A., Kwon, Y., Peter, S., and Kasikci, B. (2020), "{AGAMOTTO}: How Persistent is your Persistent Memory Application?" in *14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20)*, pp. 1047–1064.

Niu, J. (2018), "Hybrid system analysis and design for scale-out storage environments," PhD Thesis, DR-NTU, Nanyang Technological University Library, Singapore.

Niu, J., Xu, J., and Xie, L. (2018), "Hybrid Storage Systems: A Survey of Architectures and Algorithms," *IEEE ACCESS*, 6, 13385–13406.

Norman, L. (2009), "RamSan-20 PCIe Flash SSD: Expanding the role of Flash in the enterprise," .

Organick, L., Ang, S. D., Chen, Y.-J., Lopez, R., Yekhanin, S., Makarychev, K., Racz, M. Z., Kamath, G., Gopalan, P., Nguyen, B., et al. (2018), "Random access in large-scale DNA data storage," *Nature biotechnology*, 36, 242.

Qureshi, M. K., Franceschini, M. M., Jagmohan, A., and Lastras, L. A. (2012), "PreSET: improving performance of phase change memories by exploiting asymmetry in write times," *ACM SIGARCH Computer Architecture News*, 40, 380–391.

Repository, S. I. T. (2017), "Systor'17 Fujitsu Laboratory Traces," .

Roy, T. and Kant, K. (2020), "Enhancing Endurance of SSD Based high-performance Storage Systems using Emerging NVM Technologies," in *IEEE IPDPS Workshops*, pp. 1070–1079.

Roy, T., Gupta, J., Kant, K., Pal, A., Minturn, D., and Tavvakol, A. (2021), "Managing SSD tail latency with PLM," *Proc. of NAS*.

Rudoff, A. (2013), "The impact of the NVM programming model," in *Storage Developer Conference*.

Salkhordeh, R., Mutlu, O., and Asadi, H. (2019), "An Analytical Model for Performance and Lifetime Estimation of Hybrid DRAM-NVM Main Memories," *arXiv preprint arXiv:1903.10067*.

Shi, X., Li, M., Liu, W., Jin, H., Yu, C., and Chen, Y. (2017), "Ssdup: a traffic-aware ssd burst buffer for hpc systems," in *Proceedings of the international conference on supercomputing*, pp. 1–10.

Shi, X., Liu, W., He, L., Jin, H., Li, M., and Chen, Y. (2019), "Optimizing the SSD Burst Buffer by Traffic Detection," *arXiv preprint arXiv:1902.05746*.

Subedi, P., Davis, P. E., Villalobos, J. J., Rodero, I., and Parashar, M. (2018), "Using Intel Optane Devices for In-situ Data Staging in HPC Workflows," *arXiv preprint arXiv:1807.09651*.

Tai, M.-Y., Pillai, S. B., Ji, Y.-L., and Liu, H. (2018), "Memory apparatus and methods thereof for write amplification aware wear leveling," US Patent App. 15/396,548.

Tarihi, M., Asadi, H., Haghdoost, A., Arjomand, M., and Sarbazi-Azad, H. (2016), "A hybrid non-volatile cache design for solid-state drives using comprehensive I/O characterization," *IEEE Transactions on Computers*, 65, 1678–1691.

Tavakkol, A., Sadrosadati, M., Ghose, S., Kim, J., Luo, Y., Wang, Y., Ghiasi, N. M., Orosa, L., Gómez-Luna, J., and Mutlu, O. (2018a), "FLIN: Enabling fairness and enhancing performance in modern NVMe solid state drives," in *ACM/IEEE ISCA*, pp. 397–410.

Tavakkol, A., Gómez-Luna, J., Sadrosadati, M., Ghose, S., and Mutlu, O. (2018b), "MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices," in *USENIX FAST*, pp. 49–66.

Tesfay, W. B., Hofmann, P., Nakamura, T., Kiyomoto, S., and Serna, J. (2018), "PrivacyGuide: towards an implementation of the EU GDPR on internet privacy policy evaluation," in *Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics*, pp. 15–21.

Torabzadehkashi, M., Heydarigorji, A., Rezaei, S., Bobarshad, H., Alves, V., and Bagherzadeh, N. (2019), "Accelerating HPC applications using computational storage devices," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp. 1878–1885, IEEE.

Verma, A., Koller, R., Useche, L., and Rangaswami, R. (2010), "SRCMap: Energy Proportional Storage Using Dynamic Consolidation." in *FAST*, vol. 10, pp. 267–280.

Wang, C. and Wong, W.-F. (2012), "Observational wear leveling: an efficient algorithm for flash memory management," in *Proceedings of the 49th Annual Design Automation Conference*, pp. 235–242, ACM.

Weigend, A. S. (2018), *Time series prediction: forecasting the future and understanding the past*, Routledge.

Xu, Y., Musgrave, Z., Noble, B., and Bailey, M. (2013), "Bobtail: Avoiding long tails in the cloud," in *USENIX NSDI*, pp. 329–341.