PLMC: A Predictable Tail Latency Mode Coordinator for Shared NVMe SSD with Multiple Hosts

T. Roy, J. Gupta, K. Kant, A. Pal
Temple University, USA
[tanaya.roy|jit.gupta|kkant|amitangshu.pal]@temple.edu

D. Minturn
Intel Corp, USA
dave.b.minturn@intel.com

A. Tavakkol
Fortum, Switzerland
arash82ir@gmail.com

***Abstract*—Solid-State Drives (SSDs) involve a complex set of management activities in the background, resulting in unpredictable delays and occasional extended access latencies. However, there is an increasing demand for "deterministic" access latency in a growing number of scenarios. This cemand has prompted a new feature in the NVMe storage access protocol called *Predictable Latency Mode* (PLM), which provides a way to tighten tail latency in SSDs. This paper presents the first study of the PLM feature in a single-host environment and its extension to multi-host settings. 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 with a simple traffic prediction can achieve 31% improvement than without a coordinator on the 90%-tail latency values.**

I. Introduction

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 lower end-to-end storage latencies [1]–[4]. However, the tail latency in current SSDs can range up to several milliseconds. For example, the latency distribution of a recent Intel NAND SSD device (Intel DC P4610 SSD) with a workload of 70%-read and 30%-write 4KB requests experience more than 1 ms 99% tail latency as shown in Fig. 1 due to different background operations (see section II-A) and their interference with I/O accesses[1] [5]. 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 the access latency does not vary substantially from transaction to transaction [6]. Thus, tightly controlling the tail latency is essential for emerging storage systems [7]–[9]. Furthermore, high variability in storage access latency makes the resource allocations challenging when multiple applications with different quality of service (QoS) requirements share a storage volume.

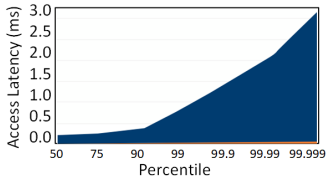

Fig. 1: Latency Distribution for Intel DC P4610 SSD

The NVMe 1.4 specification (NVMe v1.4) standard introduces the predictable latency mode (PLM) [10] 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 II-B). 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. In case all these workloads are running on the same host, the host itself can regulate its share of the DTWin budget. However, when the workloads run on different hosts, more complex coordination is required. Typical data center environment concentrates storage in a small number of "storage servers" which are accessed by all the hosts; therefore, the scenario of multiple hosts accessing data on the same SSD is common and is likely to 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. Low read latency is one of the significant challenges to achieve for these applications. This challenge is the primary motivation for extending the existing PLM feature to a multi-host environment where several workloads compete to execute within a limited DTWin budget.

The existing PLM mechanism, as currently defined, focuses only on serving the reads, and hence this paper is also focused on the reads. It calls for buffering 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 works.

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 very new and still not available in any commercial SSDs, we need to lean on simulation to examine PLM performance. For this, we have modified MQSim [11], which is a comprehensive and validated model of NVMe and SATA-based SSD. This modification provides a capability that the research community can use to study the PLM feature further.

[1]This figure is only for illustration; our experimental setups don't use this SSD and the workload is different too.

- **Extending PLM feature**: The PLM feature is currently defined only for a single host achieving deterministic IO latency. In this paper, 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.
- **PLMC Evaluation**: The proposed PLMC predicts traffic of different classes and uses it to allocate the DTWin "counts." We show that this mechanism can achieve up to 31% improvement in serving the highest QoS class compared to performing the same application without a coordinator, despite very high burstiness of the traffic.

The outline of the paper is as follows. Section II provides an in-depth discussion of the PLM feature as specified in NVMe1.4, and its limitations along with the the motivation behind the proposed PLMC. Section III discusses the detailed design of PLMC. Section IV provides our simulation and emulation results. Finally, section V concludes the discussion.

II. PLM Mechanism and Its Deficiencies

A. SSD Structure, Background Activities and Access

The NVM storage model defines several concepts including NVM subsystem, domain, endurance group, NVM-sets and namespaces, as illustrated in Fig 2. 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-Sets, which consist of one or an array of namespaces. Each endurance group is an organizational construct for wear leveling purposes.

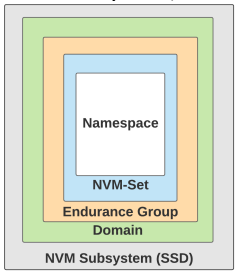

Fig. 2: NVM Storage Hierarchy

SSDs internally involve complex management activities that are performed largely in the background by the firmware known as *Flash Translation Layer* (FTL) [12], [13]. The primary role of FTL is to hide the complexities of erasure operation [14], out-of-place writes [15], address translation [16], [17], wear-leveling [18], [19], and garbage collection [11], [20]–[22]. These activities and their interference with normal read/write operations can extend the access latency from its nominal value (<100 μs) into several milliseconds [23], [24]. Fig. 1 shows 99% latency can exceed 1 ms.

The 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 [25], [26].

B. PLM Feature and Its Functioning

The PLM is enabled at NVM-set granularity and, therefore, NVM-set cycles between the aforementioned DTWin and NDWin time windows, 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 such as DTWin read typical(C_{RD}), DTWin Write typical(C_{WR}), DTWin time maximum(T_{DW}). The C_{RD} and C_{WR} are collectively considered as DTWin "counts" (DC) in this paper. DC and T_{DW} are regarded as the DTWin budget. 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. By ensuring that all NVM-sets copies are never in the NDWin state simultaneously, an application can continuously read data in the DTWin mode from one of the NVM-set copies as depicted in Fig 3. The NVM-set may transition to the NDWin before exhausting its DTWin budget – this happens if there is an emergency management operation required The amount of time spent by the NVM-set in NDWin state depends on the necessary background operations for management activity.[2]

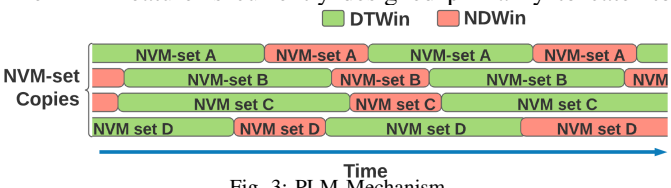

Fig. 3: PLM Mechanism

The Fig 4 represents how a remote host can communicate to an NVMe controller to exploit the PLM feature. The NVMe controller posts the DTWin related status for each NVM-set into a "log-page" of the NVM set accessible to the host using NVMe commands. NVMe controller advertises those "log-page" changes to the host. PLM's current specification puts the regular transition control with the host and the forced transition by the NVMe controller (if the host does not respect operating rules).

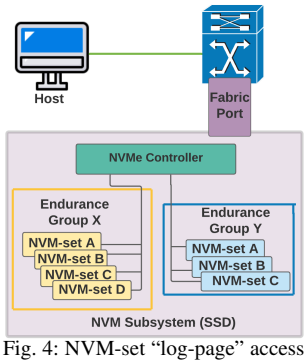

Fig. 4: NVM-set "log-page" access by a remote host

[2]Note that the duplication 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 deterministic service as well. RAID1 type SSD arrangement is somewhat different and based on the ongoing background operation on SSDs.

The host defines the beginning of DTWin and NDWin periods by making requests to the NVMe controller 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. SSDs internally involves complex management activities that are performed largely in the background by the firmware known as *Flash Translation Layer* (FTL) [12], [13]. The primary role of FTL is to hide the complexities of erasure operation [14], out-of-place writes [15], address translation [16], [17], wear-leveling [18], [19], and garbage collection [11], [20]–[22]. These activities and their interference with normal read/write operations can extend the access latency from its nominal value (<100 μs) into several milliseconds [23], [24]. Fig. 1 shows 99% latency can exceed 1 ms. Inefficient management could also reduce the endurance, i.e., the number of program-erase cycles that the SSD can endure [15], [27]–[30].

The 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 [25], [26].

C. 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 priority workloads of different QoS classes (i.e. high, medium, and low) considering few IO-intensive portions from Systor 2017 traces [31], a month-long virtual desktop infrastructure (VDI) read-intensive trace. Fig. 5 shows the read IO size of all three considered workloads, which shows a wide variations.

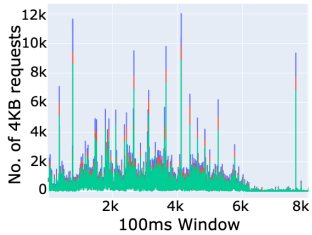

Fig. 5: Mixture of Workloads with different Priorities; Blue: High QoS, Red: Medium QoS, Green: Low QoS

We now study the impact on latency for high QoS workloads of Fig. 5 in a shared environment when multiple workloads share the same data storage resources. To do that, we have simulated PLM in the MQSim simulator with appropriate modifications (detailed in Section IV-A). We wanted to observe the PLM contribution to a workloads' experienced latency when each workload is run in isolation and in the shared environment with different QoS class workloads. In the Fig. 6, we observed that the high QoS workload obtains 3X times 90% tail latency when running with medium and low QoS workloads during a span of 15 minutes. 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.

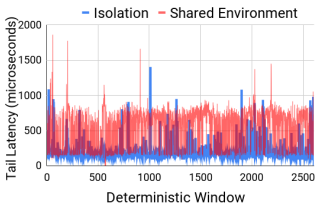

Fig. 6: Tail Latency in isolation vs sharing

III. 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 via the NVMe controller. This NVMe controller can incorporate a DC-allocate module that could help 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 NVMe controller. Therefore, the DC-allocate module needs to be outside of the NVMe controller to coordinate among different host accesses to control the access latency tail in a multi-host environment. The proposed PLMC embodies this functionality.

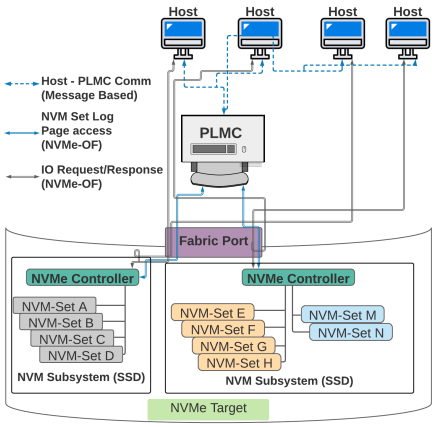

Fig. 7: Proposed PLMC Architecture

A. Coordinating Multiple Host Accesses Through PLMC

The proposed PLMC is shown in Fig. 7. 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 NVMe controller that controls a number of NVM-sets. The NVM-set copies DTWin attributes which are maintained in a log page on a per NVM-set basis. The log page can be accessed via the corresponding NVMe controller. The PLMC uses the conventional NVMe-OF (NVMe over fabric) protocol [25], [26] to communicates to NVMe controller(s) and accesses each NVM-sets' 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 PLM coordinator 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 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, the PLMC's DC allocation mechanism can consider the QoS classes of the host. The host, with the allocated DC, can perform remote IO operations via the NVMe-oF protocol.

The storage server forms an ideal place for PLMC to live. All hosts with similar traffic characteristics will experience the same underlying device access latency in the absence of management operations; therefore, the difference will be mostly in tail latency. Note that the latency of concern here is simply the device access 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).

PLMC supports multiple application classes based on the tail latency requirement. It is intended to be lightweight and can gracefully handle the uncertainties in IO accesses and the NDWin state's transition. To obtain deterministic latency, a host needs to access the device during DTWin while respecting DTWin attribute values. We assume that each host requests allocation of DC from PLMC at the start of DTWin and the allocated DC's do not change during the window. If the host receives fewer than the required counts, it can request PLMC for additional DC allocation. Several DC allocation policies (incorporated in the PLMC) for several real-world storage traces are studied in this paper.

B. Host Traffic Estimation by PLMC

The PLMC's primary role is to estimate or receive the DT count needs of each of the participating hosts and make the DC allocation accordingly. As stated earlier, storage server is a suitable place for PLMC; 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 [32], which we do not focus upon in this paper.

The key parameter required by PLMC is an estimate of the DTWin count (DC) allocated to each host for the next DT window. 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 helpful. We used a simple exponential smoothing-based prediction. Though accurate traffic prediction can enhance PLMC's performance, studying traffic prediction is not our concern in this manuscript. However, our interest is to show how PLMC can exploit a simple traffic prediction. We will comment on this aspect in section IV.

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

$$\mathcal{R}_j^{(p)}(n+1) = \zeta \mathcal{R}_j^{(m)}(n) + (1-\zeta)\mathcal{R}_j^{(p)}(n) \tag{1}$$

C. Deterministic Count Allocations by PLMC

As stated above, PLMC uses a simple exponential smoothing-based estimation of DC required by each host, which we call *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 count precisely how many requests will happen in the next window for our trace-based experiments 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 allocation policy can do if we knew the exact DC requirements in every DTWin period.

TABLE I: Notation

Symbol	Explanation
$W(n)$	Actual duration of nth scheduling Window.
$W_d(n)$	Actual duration of nth deterministic Window, which can be less than or equal to T_{DW} at nth window
$W_{nd}(n)$	Actual duration of nth non-deterministic Window such that $T_{NDWL} \leq W_{nd}(n) \leq T_{NDWL}$
C	Available DT Count 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 DT period
L_{nd}	Latency of 4KB IO in a ND period
r_j	Fixed ratio for j-th class to assign the residue DC
$\mathcal{R}_j^{(m)}(n)$	Measured Traffic (#IOs/DT-period) for j-th class during $W(n)$
$\mathcal{R}_j^{(p)}(n)$	Predicted Traffic (#IOs/DT-period) for j-th class during $W(n)$
T_{DW}	Maximum Deterministic Window Timeperiod
$u_j(n)$	Allocated DC utilization for j-th class during $W(n)$
$v_j(n)$	Deficiency of DC during $W(n)$

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* and denote it as $W(n)$. We can infer,

$$W(n) = W_d(n) + W_{nd}(n), \text{ for } n = 0, 1, ... \tag{2}$$

The complete set of notations are given in Table I. 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. $W_d(n) \leq T_{DW}$, 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:

Definition 1 (Utilization): The utilization of a host is defined by the ratio between the number of DCs consumed vs. allocated. The utilization of j-th host during $W(n)$ is given by:

$$u_j(n) = C_j^{\text{used}}(n) / C_j^{\text{alloc}}(n) \tag{3}$$

Definition 2 (Deficiency): The deficiency of a host is defined as the fraction of requests that are not covered by the allocated DC. The deficiency of j-th host during $W(n)$ is given by:

$$v_j(n) = \begin{cases} \frac{\mathcal{R}_j^m(n) - C_j^{\text{used}}(n)}{\mathcal{R}_j^m(n)} & \mathcal{R}_j^m(n) > C_j^{\text{used}}(n) \\ 0 & \text{Otherwise} \end{cases} \tag{4}$$

Definition 3 (Tail Latency): The 90% tail latency is measured for each of the considered host running workload of individual QoS class. The latency to execute each of these requests could be deterministic L_d or non-deterministic L_{nd}. Therefore, the tail latency, $L_j^{(m)}$, of the j-th host during $W(n)$ is defined as

$$L_j^{(m)}(n) = \begin{cases} L_d & \text{for } \mathcal{R}_j^{(m)}(n) \le C_j^{\text{alloc}}(n) \\ L_{nd} & \text{Otherwise} \end{cases} \tag{5}$$

where $C_j^{\text{alloc}}(n)$ denote the DC allocated for the j-th host during $W_d(n)$.

The $C_j^{\text{alloc}}(n)$ can be expressed as follows. If the total predicted traffic $\sum_{j=1}^{q} \mathcal{R}_j^{(p)}(n)$ is below C, then we allocate the predicted value directly, i.e., $C_j^{\text{alloc}}(n) = \mathcal{R}_j^{(p)}(n)$. Otherwise all the host will get their required minimum DC, $C_j^{\min}$, conveyed to PLMC. The residual DC $C^{\text{res}}(n)$ (i.e., the counts remaining after the minimum allocation) can be distributed among the q hosts based on their workloads' QoS class priority; this additional allocation corresponding to host-j is denoted as $\phi_j(n)$, i.e.

$$C_j^{\text{alloc}}(n) = \begin{cases} \mathcal{R}_j^{(p)}(n) & \text{for } \sum_{j=1}^{q} \mathcal{R}_j^{(p)}(n) \le C \\ D_j^{\min} + \phi_j(n) & \text{Otherwise} \end{cases} \tag{6}$$

To calculate $\phi_j(n)$, we use the following two methods for assigning $C^{\text{res}}(n)$ to various QoS classes:

1) **Policy I (Strict Priority):** The host running highest QoS class workload is assigned the DC required to match the predicted traffic for the current window, if $C^{\text{res}}(n)$ is enough. Otherwise, $C^{\text{res}}(n)$ is allocated to the host running workloads of next lower class, until we run out of the counts. (Obviously, the last host to get an allocation may get less than its full requirement).
2) **Policy II Fixed Ratio:** Here $C^{\text{res}}(n)$ is split among various hosts based on a predefined set of ratios r_j corresponding to the different classes of workloads running.

IV. Evaluation of the Proposed PLMC

A. Enhancing MQSim Simulator to Support PLMC

The PLM feature is relatively new, and 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. MQSim [11] attempts to build a very detailed and realistic model of SSDs. It explicitly represents the flash device characteristics and operation, the SSD's internal architecture, detailed FTL operations, device-level caching, and host interfacing. It provides an exact representation of both SATA and NVMe interfaces, of which we use NVMe differentiated queuing feature to support different QoS classes.

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 v1.4 features. Consequently, we embarked upon an extensive effort to understand the implementation and enhance it for our needs, as explained below.

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 later, 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.

For the DTWin, NDWin, and DC features, we introduced some parameters set at the start of a simulation run. These include (a) Maximum DTWin and NDWin period (in nanoseconds), after which the SSD will autonomously transition to the other state, and (b) DC value, from which we allocate counts to all the hosts. MQSim also does not have a provision for measuring the tail latency. We have included reporting 90th percentile tail latency experienced by the host when the DTWin period ends or exhausts all allocated DCs 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.

B. Emulation of PLMC on Real Server

To obtain some real experimental results on the DTWin-like mechanism using regular SSDs, we use an Intel server installed with two physical SSDs, Samsung Evo 970 SSD, so that at least one of them is always in the DTWin mode. Since it is critical to schedule SSD maintenance operations only during the NDWin period, we introduce an additional delay to emulate these operations during the NDWin period.

C. Workloads and Configurations Used

For evaluation, we used the Systor 2017 trace [31], a month-long virtual desktop infrastructure (VDI) read-intensive trace that consists of wide variations of IO sizes. We combined a few selected portions of the original trace to create a variety of workloads with distinct IO intensities. Table II lists three datasets. Each of these datasets consists of three workloads with different IO intensities(reported as average 4KB requests per DTWin in Table II), considered as three QoS classes as high, medium and low respectively.

TABLE II: 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 at the beginning of each window, a large window will reduce the effectiveness of the control.

Because of this trade-off, we have used DC and DTWin period (T_{DW}) configurations as represented in Table III. Here M_{DC} is the average number of DC required by all three workloads and is represented as M_{DC}. We have determined the value of M_{DC} 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 traffic variability.

TABLE III: 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

D. Evaluation Results

For evaluation, we have considered three hosts running three workloads of high, medium and low QoS classes. We evaluate here how DC of two NVM-sets copies (two SSDs) are distributed amongst three hosts based on the three metrics considered in *Definition 1,2* and *3*, namely (DT count) utilization, (DT count) deficiency, and resulting 90% tail latency respectively.

We used the same set of workloads and other settings for both simulation and emulation to compare the results. Our validation runs show that we can achieve agreement between the simulation and emulation concerning utilization and deficiency. However, the tail latencies are not comparable because of numerous differences between the simulation and emulation platforms. In particular, the simulation model uses a highly detailed model of a real SSD (along with NVMe protocol latencies) from several years ago, which we did not perturb. In contrast, the emulation uses a contemporary, much lower latency, NVMe SSD. Also, because of the request tagging difficulties on the real SSD, we could not use different NVMe queuing priorities in the emulation. Therefore, we will not compare tail latencies across the two cases.

Fig 8 shows the overall results for dataset1. Similar results were obtained for datasets 2 and 3 and, therefore, not reported. The top part of the figure shows results for the emulation and the bottom portion for the simulation. The three hosts running three QoS class workloads are represented as "High", "Medium" and "Low" respectively in the table. We have listed separate cases for strict Priority and Fixed Ratio in each case, and under those for coordinator directed (CoD) and host-directed (HoD) cases. Even though Table III shows four different configurations, to avoid clutter, we report results only for configurations 2 and 4. The figure reports on utilization, deficiency and tail latency measures for High, Medium, and Low QoS classes, which we discuss next. We generated 10 percentile, 50 percentile (Median), and 90 percentile values for each measure. However, not all are listed in the figures for brevity. The reason to choose Median, instead of Mean, is that it is less affected by occasional large quantities, which are common due to very bursty nature of the traffic.

Utilization: In Table III we only report 10 percentile utilization. The others (50 & 90%) are not reported since they all turn out to be 100% in all the cases! A 100% utilization indicates that we are not wasting any allocated counts, a sign of scarce resources. Note that even 10% utilization numbers are 1 for the HoD case because we know exactly what is needed in this case. The CoD numbers are much lower because the PLMC does not know the precise needs and therefore has to be generous in its estimation. The key point to note is an extremely close agreement across simulation and emulation. In the table, another interesting piece is the "Utilization" value reported for each host over the deterministic windows for strict priority and fixed ratio, respectively, under CoD and HoD. We

EMULATION																	
Approaches	Treatment	Configuration	Utilization			Deficiency						Tail Latency					
			10%			Median (50%)			90%			Median (50%)			90%		
			High	Medium	Low	High	Medium	Low	High	Medium	Low	High	Medium	Low	High	Medium	Low
Strict Priority	Coordinator Directed	2	0.55	0.56	0.57	0.03	0.04	0.05	0.25	0.27	0.41	52	51	52	2241	2268.6	2300
		4	0.84	0.86	0.93	0.03	0.05	0.05	0.25	0.29	0.34	48	52	903	2480	2480	4539
	Host Directed	2	1	1	1	0	0.08	0.13	0.63	0.64	0.64	14	14	35	86	54	627649.7
		4	1	1	1	0.02	0.5	0.5	0.5	0.5	0.5	45	937.5	1165	1301	1476.9	1487.3
Fixed Ratio	Coordinator Directed	2	0.55	0.57	0.57	0.01	0.05	0.23	0.14	0.20	0.74	53	52	52	2245	2272.1	2275
		4	0.84	0.87	0.91	0.00	0.12	0.16	0.14	0.22	0.30	48	103	295	2432	2349	3039
	Host Directed	2	1	1	1	0	0	0	0.66	0.66	0.66	17	24	50	53	186.8	625717.6
		4	1	1	1	0.001	0.16	0.24	0.35	0.35	0.37	27	21	51	816	756	623909
SIMULATION																	
Approaches	Treatment	Configuration	Utilization			Deficiency						Tail Latency					
			10%			Median (50%)			90%			Median (50%)			90%		
			High	Medium	Low	High	Medium	Low	High	Medium	Low	High	Medium	Low	High	Medium	Low
Strict Priority	Coordinator Directed	2	0.56	0.56	0.56	0.001	0.03	0.05	0.28	0.35	0.33	166.85	269.59	2127.13	548.17	773.36	6589.41
		4	0.87	0.88	0.89	0	0.02	0.04	0.28	0.49	0.49	176.50	290.19	2637.62	575.94	969.68	6656.05
	Host Directed	2	1	1	1	0	0	0	0	0.52	0.54	161.54	245.16	2755.19	417.19	924.61	7155.31
		4	1	1	1	0	0	0	0	0.54	0.55	174.82	284.33	2910.05	441.30	1055.67	6733.85
Fixed Ratio	Coordinator Directed	2	0.54	0.56	0.56	0	0.11	0.11	0.15	0.35	0.36	165.74	269.10	2131.03	549.84	763.49	6578.00
		4	0.84	0.88	0.89	0	0.27	0.28	0.15	0.54	0.54	176.50	287.71	2581.62	564.97	991.93	6638.08
	Host Directed	2	1	1	1	0	0	0	0	0.39	0.42	164.21	247.31	2633.16	437.99	935.46	7000.05
		4	1	1	1	0	0	0	0	0	0	176.50	291.69	2768.12	475.00	1062.36	6648.64

Fig. 8: Evaluation results for Dataset 1. The tail latency is calculated in microseconds.

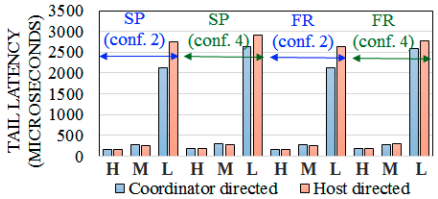

Fig. 9: Comparison of Median tail latencies for Dataset 1.

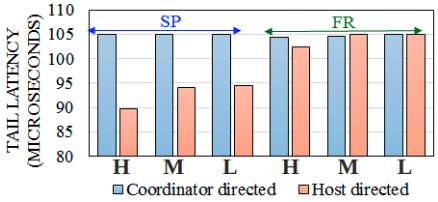

Fig. 10: Comparison of Median tail latencies for Dataset 4.

see a good agreement between the two. This figure shows the excess DT count allocation resulting from an overestimation of the traffic results in less than 100% utilization. All three hosts experience roughly the same utilization level, with the "Low" dominating slightly. The simple reason behind this is "Low" allocated less DC compared to the traffic experienced and hence, used up all DC allocated.

Deficiency: For deficiency, we ignore the 10% values but show 50% and 90% in Table III. It is seen that the median deficiencies are rather small: for HoD, we expect them to be small, but for CoD, they are small since the available counts are significantly higher than the average. Moreover, there is an excellent agreement between emulation and simulation results reported in the table. As expected, the 90% numbers are much more variable, and the agreement between emulation and simulation is somewhat worse but still quite good overall. It is worth noting that both emulation and simulation data are based on about 9000 DT Windows for configuration 2 (and 2250 for configuration 4), which means that 90 percentile values are expected to be rather variable.

Tail Latency: For latency as well, we report 50% and 90% values for 90% Tail Latency. We computed the 90% Tail Latency during each DTWin and then reported the top 50%, and 90% values experienced during the entire workload. As stated before, the latencies are not comparable across emulation and simulation, and we shall not attempt to do so. Also, since the simulation model is far more granular here, we subsequently comment only on simulation results. The median values are shown more clearly in Fig. 9 (High, Medium and Low QoS traffic are referred to as *H*, *M* and *L* respectively in Figs. 9 and 10). It is seen that there is excellent agreement between HoD and CoD values across all three QoS classes and for both strict priority and fixed ratio cases. This is remarkable in view of extremely simple exponential smoothing-based prediction and the highly bursty nature of the traffic. It is also worth noticing that the latencies of High and Medium classes are controlled quite well at the cost of Low QoS class. The dataset DS1 is as same as reported in Fig 5. If we compare the latency experienced by "High" in shared environment Fig 6 with the PLMC reported latency, we have achieved 31% improvement.

Looking at the 90% tail latency, we again find good control over High and Medium QoS classes' latency, with a significantly lower latency for the High QoS than Medium QoS. However, when we look at the 90% tail latency values, the agreement becomes poorer. For example, for config 2 and strict priority, the CoD values are (548, 773, 6589) whereas the HoD values are (417, 924, 7155). The tail latency is most important for "High", where the CoD is about 31% higher. Again, such accuracy is quite good, considering the stress case considered here: very high burstiness and synchronized traffic. The direction of error reverses for the Medium priority: poorer treatment of high priority will result in bbetter treatpriorities. Finally, there is a decent agreement on the "Low", but latency is not essential for that class. One area of concern that emerges from our evaluation is an estimation of DT count needs by the PLMC, as discussed earlier. Since time-series methods are unlikely to be helpful, a machine learning-based traffic predictor is another possible approach. However, even a complex neural net is unlikely to work well unless the traffic does have significant repeating patterns.

Given this, we believe that a hinting mechanism by the hosts

(e.g., by the applications running on the hosts) could be a practical and helpful method. However, the details of such an approach are beyond the scope of this paper.

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. 10 shows the variation of tail latency for Dataset4. It is seen that the latency drops drastically, by almost 30X, as compared to Fig. 9. 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.

V. 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 define a new entity called PLM controller (PLMC) that runs on a storage server and interacts with all the hosts using that storage server. The PLMC appears as another host to NVMe devices and can be implemented without any changes to the current NVMe v1.4 standard. We evaluated the proposed PLMC in two ways: making changes to a comprehensive SSD simulator (MQSim) and using real implementation on a server with identical SSDs to achieve comparable DTWin periods. 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 31% improvement in 90 percentile tail latency.

The PLMC could benefit from improved traffic prediction, but more sophisticated time-series prediction methods do not seem beneficial because of the high burstiness of the storage traffic. Instead, a hinting mechanism from hosts (i.e., from applications running on the hosts) to PLMC could be more helpful. We will explore this aspect in the 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 it switch-over to the nondeterministic mode.

References

[1] R. Micheloni *et al.*, *Inside solid state drives (SSDs)*. Springer, 2013.

[2] D. Cobb *et al.*, "Nvm express and the pci express ssd revolution," 2012, intel Developer Forum.

[3] H. Strass, "An introduction to nvme," https://www.seagate.com/files/www-content/product-content/ssd-fam/nvme-ssd/nytro-xf1440-ssd/shared/docs/an-introduction-to-nvme-tp690-1-1605us.pdf.

[4] A. Huffman, "Nvm express," *revision 1.0 c. Intel Corporation*, 2012.

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

[6] S. Chen *et al.*, "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*, 2019, pp. 107–120.

[7] Y. Xu *et al.*, "Bobtail: Avoiding long tails in the cloud," in *USENIX NSDI*, 2013, pp. 329–341.

[8] G. DeCandia *et al.*, "Dynamo: Amazon's highly available key-value store," *ACM SIGOPS operating systems review*, vol. 41, no. 6, pp. 205–220, 2007.

[9] J. Dean *et al.*, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[10] "Nvm express base specification, rev 1.4," https://nvmexpress.org/wp-content/uploads/NVM-Express-1_4-2019.06.10-Ratified.pdf, June 2019.

[11] A. Tavakkol *et al.*, "Mqsim: A framework for enabling realistic studies of modern multi-queue SSD devices," in *USENIX FAST*, 2018, pp. 49–66.

[12] K. Eshghi *et al.*, "Ssd architecture and pci express interface," in *Inside solid state drives (SSDs)*. Springer, 2013, pp. 19–45.

[13] R. Micheloni *et al.*, "Solid state drives (ssds)," in *Solid-State-Drives (SSDs) Modeling*. Springer, 2017, pp. 1–17.

[14] P. Bednar *et al.*, "Ssd: New challenges for digital forensics," in *ItAIS*, 2011.

[15] M. Abraham *et al.*, "Nand flash trends for ssd/enterprise," *Flash Memory Summit*, 2010.

[16] D. Ma *et al.*, "A survey of address translation technologies for flash memories," *ACM Computing Surveys (CSUR)*, vol. 46, no. 3, pp. 1–39, 2014.

[17] D. Liu *et al.*, "Durable address translation in pcm-based flash storage systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 2, pp. 475–490, 2016.

[18] A. I. Alsalibi *et al.*, "A survey of techniques for architecting slc/mlc/tlc hybrid flash memory–based ssds," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 13, p. e4420, 2018.

[19] S. Kim *et al.*, "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*, vol. 24, no. 3, pp. 1–9, 2019.

[20] Y. Cai *et al.*, "Error characterization, mitigation, and recovery in flash-memory-based solid-state drives," *Proceedings of the IEEE*, vol. 105, no. 9, pp. 1666–1704, 2017.

[21] L.-P. Chang *et al.*, "Real-time garbage collection for flash-memory storage systems of real-time embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 4, pp. 837–863, 2004.

[22] E. Gal *et al.*, "Algorithms and data structures for flash memories," *ACM Computing Surveys (CSUR)*, vol. 37, no. 2, pp. 138–163, 2005.

[23] J. Kim *et al.*, "Alleviating garbage collection interference through spatial separation in all flash arrays," in *USENIX ATC*, 2019, pp. 799–812.

[24] A. Tavakkol *et al.*, "Flin: Enabling fairness and enhancing performance in modern nvme solid state drives," in *ACM/IEEE ISCA*, 2018, pp. 397–410.

[25] D. Minturn *et al.*, "Under the hood with nvme over fabrics," in *Ethernet Storage Forum. SNIA*, 2015.

[26] Z. Guz *et al.*, "Nvme-over-fabrics performance characterization and the path to low-overhead flash disaggregation," in *ACM SYSTOR*, 2017, pp. 16:1–16:9.

[27] V. Mohan *et al.*, "How i learned to stop worrying and love flash endurance." *HotStorage*, vol. 10, pp. 3–3, 2010.

[28] Y. Hu *et al.*, "Performance impact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity," in *ICS*, 2011, pp. 96–107.

[29] T. Roy *et al.*, "Enhancing endurance of ssd based high-performance storage systems using emerging nvm technologies," in *IEEE IPDPS Workshops*, 2020, pp. 1070–1079.

[30] N. Agrawal *et al.*, "Design tradeoffs for ssd performance." in *USENIX Annual Technical Conference*, vol. 57. Boston, USA, 2008.

[31] S. I. T. Repository, "Systor'17 fujitsu laboratory traces." [Online]. Available: http://iotta.snia.org/traces/4964

[32] F. S. Gharehchopogh *et al.*, "A survey and taxonomy of leader election algorithms in distributed systems," *Indian Journal of Science and Technology*, vol. 7, no. 6, p. 815, 2014.