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

Tanaya Roy*, Jit Gupta*, Krishna Kant*, Amitangshu Pal†, Dave Minturn‡, Arash Tavakkol§
*Computer and Information Sciences, Temple University, Philadelphia, PA 19122, USA
†Computer Science and Engineering, Indian Institute of Technology Kanpur, Kanpur, India
‡Intel Corp, Hillsboro, OR 97291, USA
§Fortum, Switzerland

2021 IEEE International Conference on Networking, Architecture and Storage (NAS) | 978-1-7281-7744-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/NAS51552.2021.9605470

***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 demand 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 can achieve 82% improvement in 99.99% tail latency compared to a bare SSD without PLM feature. Moreover, the proposed coordinator with simple traffic prediction can perform 93.2% better than without coordinator on the 99%-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 low end-to-end storage latencies [1]–[4]. However, the tail latency in current SSDs can range up to several milliseconds.

For example, Fig. 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 [5]. It shows that the 99 percentile tail latency can be more than 1 ms.[1] 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 [6]. Thus, tightly controlling the tail latency is essential for emerging storage systems [7]–[9].

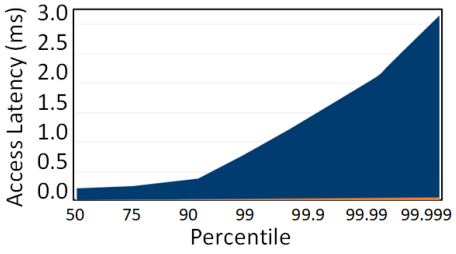


Fig. 1: Latency Distribution for Intel DC P4610 SSD

The NVMe 1.4 specification (NVMe v1.4) 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.

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 paper, we design and evaluate such a coordinator, henceforth called PLM coordinator (PLMC).

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 [11], which is a com-

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

978-1-7281-7744-1/21/$31.00 ©2021 IEEE

prehensive 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 paper, we extend it to multiple hosts sharing a 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.

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 results. Finally, section V concludes the discussion.

## II. PLM Mechanism and Its Deficiencies

### A. SSD Structure, Background Activities and Access

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 [12]–[14]. These activities and their interference with normal read/write operations can extend the access latency from its nominal value ($<$100 $\mu$s) into several milliseconds [15] as illustrated in Fig. 1.

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.

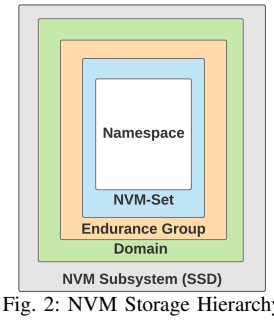

Fig. 2: NVM Storage Hierarchy

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 [16].

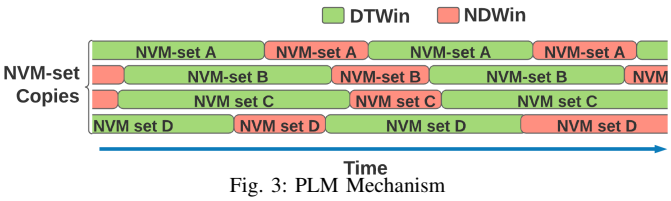

Fig. 3: PLM Mechanism

The PLM is enabled at NVM-set granularity and, therefore, NVM-set cycles between the aforementioned DTWin and NDWin time windows as fig 3 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 ($T_{DW}$). We consider the parameters $C_{RD}$ and $C_{WR}$ as DTWin "counts" (DC), and $DC/T_{DW}$ as the DTWin budgets.

### B. 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

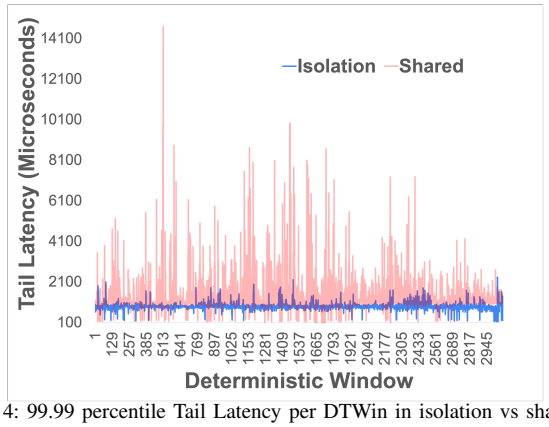


Fig. 4: 99.99 percentile Tail Latency per DTWin in isolation vs sharing

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).

### 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 workloads of different QoS classes (i.e. high, medium, and low) considering a few IO-intensive portions from Systor 2017 traces [17], 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. 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.

## 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 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 [18], which we do not focus upon in this paper.

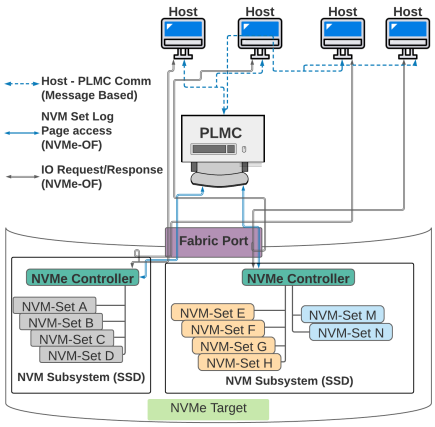


Fig. 5: Proposed PLMC Architecture

### A. Coordinating Multiple Host Accesses Through PLMC

The proposed PLMC is shown in Fig. 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 [16], [19] 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, 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 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. 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, their usefulness was questionable, and we indeed found this to be the case by implementing them. We are not reporting those results since they did not perform any better than the simple exponential smoothing that we do use for prediction. Although sophisticated methods such as those based on machine learning could potentially yield better results, that aspect is beyond the scope of this paper.

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,

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

### *C. 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. $W_d(n) \leq T_{DW}$, 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 and use them as evaluation metrices.

**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 (same as host in this paper).

The DC allocation for each host during a given $W_d(n)$ is 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 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.

## IV. Evaluation of the Proposed PLMC

### *A. Enhancing MQSim Simulator to Support PLMC*

Due to lack of commerical availability of SSDs with PLM feature, we evaluate PLMC using a comprehensive simulation model of SSDs. For this, we built the PLMC capabilities in an existing comprehensive SSD simulator called MQSim [11]. 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 has certain limitations and does not contain all the features necessary for building the PLM. It does not support the concept of NVM-sets and other NVMe v1.4 features. Consequently, we embarked upon an extensive effort to understand the implementation and enhance it for our needs. PLM simulation requires multiple host and multiple NVM-sets. MQSim can handle multiple IO flow and we consider each IO flow as host. We emulated multiple NVM-sets by virtualizing the entire address space into logical address ranges pertaining to the the number of required devices. Proceeding to the PLM features, we first had to introduce the notion of IO Determinism by stalling garbage collection procedure in MQSim and carry them out at a later time (during NDWin period).

### B. Workloads and Configurations Used

For evaluation, we used the Systor 2017 trace [17], 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 I (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 I: 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 II: 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 6. 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.

### C. Evaluation Results

For evaluation, we considered three hosts, each running workloads of high, medium and low QoS classes respectively.

| 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 |

Fig. 6: Evaluation result for CoD based 10 percentile Utilization, 50 and 90 percentile Deficiency considering DS1.


Fig. 7: 99.99 Percentile Tail Latency for HoD Simulation

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. 6 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 6. 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 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 6. 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.

**Tail Latency:** For tail latency, we report 99.99 percentile values in Fig. 7 and 8 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


Fig. 8: 99.99 Percentile Tail Latency for CoD Simulation

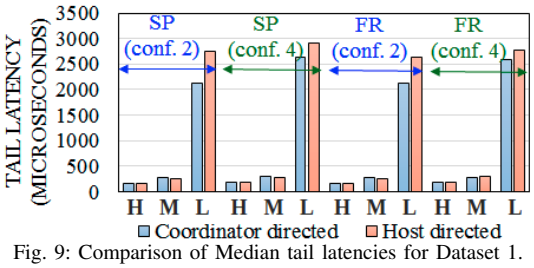


Fig. 9: Comparison of Median tail latencies for Dataset 1.

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).

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 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

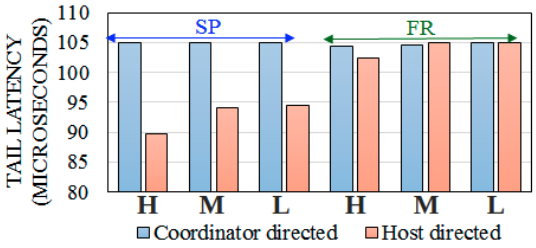


Fig. 10: Comparison of Median tail latencies for Dataset 4.

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.

## 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] 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.

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

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

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

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

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

[18] 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.

[19] 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.