IO patterns-aware and dynamic scheduling based data placement in hybrid storage system

Lei Yan*†§, Wenguo Liu*, Xuesheng Li†, Nan Su‡
Haijun Zhang†, Zaigui Zhang†‡, Dong Zhang*†
*Inspur Group Company Limited, Jinan, Shandong, 250102
‡Inspur (Zhengzhou) Data Technology Co., Ltd, Zhengzhou, Henan, 450047 China
†Inspur (Jinan) Data Technology Co., Ltd, Jinan, Shandong, 250101 China
§yanlei02@inspur.com

***Abstract*—QLC flash offers higher storage density, significantly reducing the cost of storage systems. However, the I/O performance of QLC SSD is not satisfactory, due to the limited write performance. To make matters worse, the frequent garbage collection can lead to the serious write amplification which has negative impact on the device lifespan. Traditional LRU (Least Recently Uesd) method may relieve the mentioned problem, but it comes at the cost of limited memory space. In this paper, we propose a reinforcement learning (RF) based data placement strategy for the hybrid storage system, including SLC and QLC SSDs. First, we investigate the read and write features of the different flash SSDs. Then, the IO is classified and distributed to different storage devices with a dynamic size threshold. And then, a RF-based method is proposed to extract and predict the the I/O patterns of application. Finally, the data placement strategy is established with considering the system state and the workload features. The real workload is used to evaluate the effectiveness of the proposed method, and the result shows that the access latency can be reduced by up to 40% compared with the three other similar methods.**

***Index Terms*—Hybrid storage system, Reinforcement learning, Data placement, Request scheduling, SSD**

I. Introduction

QLC (Quad-Level Cell) flash memory stores 4 bits of data per cell, offering higher storage density and lower cost, making it suitable for large-capacity storage applications. Due to the increased number of bits stored, QLC has lower durability and write performance compared to TLC and MLC, making it ideal for workloads with frequent reads but relatively fewer writes, like cold data storage.

However, the high storage density of QLC flash conflicts with the frequent write access patterns of modern applications, such as, AI training and realtime analysis. In order to improve the I/O (input and output) performance under various access patterns, researchers have proposed various effective methods which can be categorized into caching and tiering.

In Fig.1, the hybrid storage system (HSS) is composed of SLC SSD (faster) and QLC SSD (slower), and the blocks in both SSDs refer to the data with the different access frequency.

A. Caching and tiering

Data Caching refers to the temporary storage of frequently accessed or recently used data in faster memory, such as DRAM or SSDs, to minimize latency and enhance performance. It operates by keeping a subset of data closer to the CPU or application layer, enabling quick retrieval. Algorithms like LRU determine which data to retain in the cache, making it ideal for scenarios requiring immediate data access, such as database queries or real-time analytics.In data caching of Fig.1(a), and the hot data of blue and orange blocks are cached into faster SLC SSD in data access.

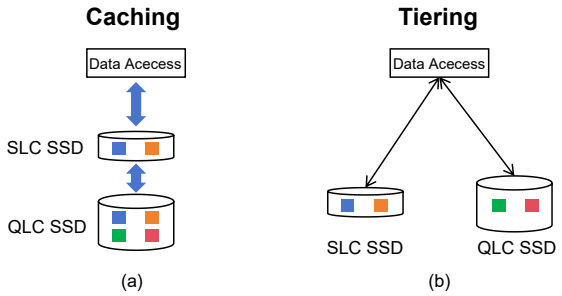

Fig. 1. Caching and tiering

Data Tiering is the practice of organizing and storing data across different storage media based on access patterns and storage characteristics. Frequently accessed ”hot” data is placed on faster, more expensive storage (e.g., SSDs), while infrequently accessed ”cold” data is stored on slower, more economical options (e.g., HDDs). This method prioritizes cost efficiency and performance by matching storage resources to data needs over time, particularly in large-scale enterprise or cloud environments. While, in Fig.1(b), the hot data is migrated into SLC, and the cold data stays in QLC.

Data caching and data tiering are complementary techniques. Caching focuses on accelerating short-term access by temporarily storing hot data, while tiering optimizes long-term storage efficiency by dynamically classifying and migrating data. Together, they enhance HSS by combining rapid access with cost-effective storage management.

B. Data placement and SSD specification

According the specification and comparison between SLC and QLC in part A, there are three facts should be considered in the design of data placment strategy.

1) The read performance of QLC SSD is not significantly different from that of SLC SSD, but QLC SSD has larger capacity, making it more suitable for read operations;

2) The write performance of SLC SCM is much higher than that of QLC SSD in terms of both bandwidth and IOPS, so SLC SCM is more suitable for write operations;
3) The flushing speed and capacity of SLC SSD need to be considered to avoid slow flushing and data loss.

Therefore, SLC is usually used as the performance tier in the HSS, and QLC as the capacity tier. The performance tier primarily handles small data write requests, while large data writes can have their placement configured based on the actual conditions of the storage system. The capacity tier mainly handles read requests, but before data is flushed from the performance tier to the capacity tier, read requests can still be served by the performance tier.

For the allocation of read and write capacity in the performance tier, factors such as write speed, flushing speed, the proportion of reads from the high-performance tier, and the proportion of reads from the capacity tier are considered when allocating the capacity for read and write data in the high-performance tier.

In this paper, our research filed is focus on the data-tiering in the hybrid flash storage system. In addition, we aim to design a novel adaptive tiering framework for the hybrid SLC+QLC storage based on the Reinforcement-learning and the I/O rescheduling. Our contributions are,

1) We investigate and analyze the read and write performance of SLC and QLC SSD, and conclude that the appropriate scenarios of QLC SSD;
2) A novel data placement strategy is designed with access pattern aware for the HSS with SLC and QLC SSDs;
3) A reinforcement learning based method is designed and adopted into the strategy establishment;
4) We implement the proposed method in a real HSS environment, and evaluate its efficiency with real and open-sourced workloads.

The rest of paper is organized as follows. Related works and their data placement models are presented in the Section II. In Section III, we describe the motivation of this paper and formulate the global latency optimization problem. In Section IV, we present IO patterns-aware rescheduling based data placement method and the reinforcement learning based strategy. The simulation results will be presented and analyzed in Section V. Finally, we summarize the paper in Section VI.

II. Related work

In the hierarchical memory, Song *et al.* [1] design novel architecture with SRAM and racetrack memory, and a data placement scheme and an instruction scheduling strategy are proposed to reduce the operation shift.

Sybil has designed a data placement module on the host side that leverages reinforcement learning to gather information about workload and HSS system configuration changes. It formulates optimal data placement strategies for various workloads and HSS systems to achieve the desired optimization goals in terms of request latency and IOPS [2].

NHC has designed a request scheduling strategy based on the read/write latency and IOPS characteristics of NAND SSDs and Optane SSDs, aiming to improve request response efficiency [3]. Specifically, it schedules additional requests from the performance tier to the capacity tier, thereby further enhancing IOPS.

In order to achieve the better performance and lower storage cost, Raina *et al.* [4] design a novel key-value store which can efficiently migrate and compact data between 3D XPoint and QLC NAND.

Under edge-cloud scenario, if the data is not placed properly, the multi-sourced data with different properties causes high file access latency. In order to reduce the access latency, Ren *et al.* [5] propose the a machine learning based data placement mechanism for HSS composed of HDD, SSD and PCM.

For SLC+QLC hybrid controller design, a modeling framework is developed to estimate the performance and endurance, which are affected by the variable write features and space utilization [6]. In addition, the framework can make the data movement strategy with the accurate and fast prediction.

Luo *et al.* [7] investigate the performance and reliability in the real hybrid SSDs, and present HyFlex, which is composed with data placement/ GC aware capacity tuning and disturb-aware data migration to deal with the QLC read disturb, performance collapse and fluctuation.

Wang *et al.* [8] present a data caching framework for the HSS with SMR and Flash SSD. The tradeoff relationship between data popularity and SMR write amplification is optimized for the global system access latency performance.

Most of the current work mentioned above focus on the data placement strategy between DRAM and SCM/SSD [1], [3], [5], [8], and there is only few work on the all flash HSS. As for the [2] and [6], the data placement strategy establishment totally relies on the IO access pattern training and prediction, and this leads to a complex IO scheduling processing.

III. Motivation and problem formulation

A. Motivation: I/O pattern awareness and rescheduling

IO pattern awareness plays a key role in formulating data placement strategies in HSS. By analyzing the IO patterns of different types of storage media (such as QLC SSD and SLC SSD), the system can choose appropriate data storage locations based on the frequency of read/write operations, data access patterns, and bandwidth and latency requirements. For example, data with high write frequency can be preferentially stored in SLC SSD, which has superior write performance, while big data size with low read frequency is better suited for the larger capacity of QLC SSD. Additionally, IO pattern awareness helps dynamically adjust data migration strategies, balancing performance and endurance, to achieve optimal resource utilization and performance optimization in multi-tier storage architectures.

Request scheduling is also important in data placement strategies establishment. By implementing effective request scheduling, the system can optimize data storage locations and access efficiency based on the performance and access patterns of different storage media. For example, in high-concurrency

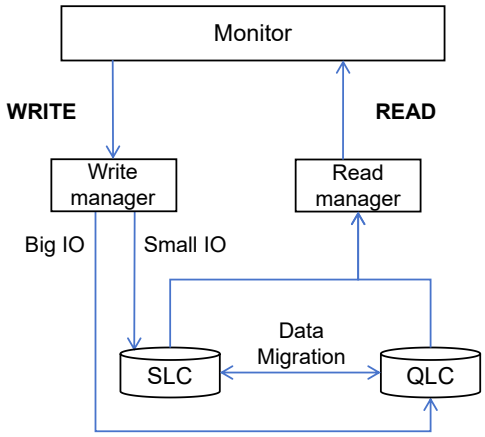

Fig. 2. I/O pattern awareness and rescheduling

read/write scenarios, the system can prioritize placing frequently accessed data on high-performance storage media, such as SLC SSD, while storing less frequently accessed data on larger capacity media, like QLC SSD. Additionally, dynamic scheduling strategies can adjust data migration and access paths based on real-time load changes, effectively reducing latency and enhancing overall system performance. Furthermore, well-designed request scheduling helps balance the load on storage media, prolonging device lifespan and further optimizing resource utilization. This dynamic adaptability enables HSS to maintain efficient and stable performance across diverse application demands.

Fig. 2 shows the framework of the data placement strategy based on I/O pattern awareness and rescheduling. The framework is composed of a monitor, a write manager and a read manager, a SLC SSD and a QLC SSD. In which, the monitor operates a reinforcement learning based method to train and predict the access patterns for the workload. The write manager recognize the IO size based on a dynamic threshold, and the read manager schedules the read IO with consideration of the resource utilization of storage space, bandwidth, and working load. The SLC and QLC SSDs are regarded as the performance layer and capacity layer.

For a write-heavy, read-many scenario: data is written to the performance tier, and when performance tier reaches its capacity threshold or meets conditions that require flushing, the data is flushed to the capacity tier. If the write operation in the performance tier is complete but the data has not yet been flushed to the capacity tier, read operations will access the data from the performance tier. Once the data has been flushed to the capacity tier but not yet deleted from the performance tier, the capacity tier is preferred for serving read requests.

For a multi-write scenario: data is written to the performance tier, with a strategy and threshold designed for flushing data to the capacity tier. When the threshold is exceeded or certain conditions are met, the data is flushed from the performance tier to the capacity tier. The challenge here is to design a mechanism for marking data as dirty in the performance tier, along with a flushing mechanism and a data consistency mechanism.

TABLE I
NOTATIONS

Notation	Meaning
$\mathcal{C}$	total IO requests number of workload
a	read-to-write ratio
$\mathcal{L}$	the total access latency
$\mathcal{A}$	the average data size for read requests
$\mathcal{B}$	the average data size for write requests
r_1, r_2	the sequential read bandwidth of SLC and QLC SSDs
w_1, w_2	the sequential write bandwidth of SLC and QLC SSDS
r_1', r_2'	4KB file random read performance of SLC and QLC SSDs
w_1', w_2'	4KB file random write performance of SLC and QLC SSDs
s_1	available space of SLC
s_2	available space of QLC

B. Problem formulation

This part aims to formulate the problem for the I/O pattern awareness and rescheduling based data placement in the HSS.

Assuming a real-world application has a total number of IO requests represented by $\mathcal{C}$, and the total access latency is $\mathcal{L}$. The read-to-write ratio is a and $1-a$, with the average data size for read and write requests being $\mathcal{A}$ and $\mathcal{B}$, respectively. Furthermore, assume that the sequential read and write bandwidth of SLC is r_1 and w_1, while for QLC it is r_2 and w_2; similarly, the 4KB file random read and write performance for SLC is r_1' and w_1', and for QLC it is r_2' and w_2'. Our goal is to minimize the overall access latency $\mathcal{L}$ by recognizing the read/write characteristics and placing the data appropriately, while scheduling access requests efficiently between hybrid storage devices. The optimization problem is defined as follows:

$$min(\mathcal{L}) = f(\mathcal{C}, a, \mathcal{A}, \mathcal{B}, r_1, w_1, r_2, w_2, r_1', w_1', r_2', w_2', s_1, s_2)$$

Moreover, based on the analysis of SLC and QLC performance and their application scenarios in Section 2, we introduce several constraints and assumptions to simplify the optimization problem and reduce its complexity:

1) Since SLC and QLC have similar read performance, we assume $r_1 = r_2$;
2) Due to the significant difference in write performance between SLC and QLC, we set all large data write requests to be handled by QLC, while small write IOs are handled by SLC;
3) To reduce design complexity, we assume data access has locality characteristics over time, which can be extracted using machine learning methods. Therefore, requests are sensed and scheduled only when there is a noticeable change in request characteristics.

Finally, the simplified optimization problem is rewritten as:

$$min(\mathcal{L}) = f(\mathcal{C}, a, r_1, w_1, w_2, s_1, s_2)$$

IV. IO PATTERNS-AWARE AND RESCHEDULING BASED DATA PLACEMENT METHOD

In this section, we first present a reinforcement learning based method for the IO patterns extraction and training.

Then, we propose a request rescheduling method based a self-adaptive threshold with the awareness of system resource. At the end, the algorithm for the data placement method is presented in Algorithm I.

A. IO patterns-aware and dynamic scheduling based data placement

In this section, we design an IO patterns-aware and dynamic scheduling based data placement method for the applications with various IO patterns. We first analyze the impact of IO patterns on the performance of HSS, and a threshold-based IO scheduling method is designed for the IO with the too large or too small request size. For the IO with other patterns, we then present a reinforcement learning based placement strategy to optimize the IO latency and system performance.

B. IO pattern analysis and request scheduling

In this paper, the direct-access model is adopted to evaluate the performance of the HSS, where all IO is directly responded to by the storage layer. In which, the HSS in this scheme is composed of the SLC SSD from Kaixia's FL6 series and QLC SSD self-developed by Inspur. In addition, the baseline performance of sequential write and read, random write and read for these two SSDs is tested, and the results are shown in table II of Section V.

We mainly analyze the impact of IO features on the HSS we have constructed, in order to select appropriate IO features for data placement strategies. In practical, the IO patterns related to the performance of the storage layer mainly include request type, request size, read-write ratio, garbage collection, and wear level. In addition, the average number of accesses and unique requests also have a certain impact on the IO efficiency of the storage system. Therefore, the IO patterns observation in our design mainly follow the following principles:

1) Observing the frequency of write operations and IO size, by scheduling write requests reasonably, can not only reduce data movement, but also reduce QLC write times, reduce SSD wear, and improve device lifespan;
2) By observing the data locality characteristics of read operations and utilizing the performance differences between SLC and QLC, data can be placed and migrated reasonably to improve system IO performance, such as read bandwidth and latency.

With above two principles, we select several key patterns with significant impact on the performance, such as request size, request type, access count of request and the location of the requested page. In addition, the remaining capacities of the SLC and QLC are observed to improve the usage efficiency of the resource.

C. Threshold based IO scheduling

With the observed IO patterns, we design a threshold based method to schedule IO dynamically. In this part, we make the IO scheduling strategies with respect of the IO types, such as READ and WRITE. In general, the IO is categorized as small if the IO size is smaller than 4KB, and the size of large IO is bigger than 4MB. For the write IO, request size, request type, access count of request and the location of the requested page

For the write IO scheduling

1) For the write IO larger than 4MB, the request is scheduled into QLC SSD to improve the IO efficiency and save the cache space;
2) For the write IO smaller than 4KB, the request is responded by SLC SSD as the cache to enhance the IO performance and reduce erase count in QLC SSD.

For the read IO scheduling

1) Due to the small read performance between QLC and SLC SSD, the read request larger than 4MB is scheduled into SLC or QLC SSD where the data belongs to;
2) For read IO smaller than 4KB, the request is responded by the data owner device from the start. If the data owner is the QLC SSD, the access count is recorded and compared with the preset threshold. If the access count during certain period is larger than the threshold, the data is mitigated to the SLC to improve the performance.

D. Reinforcement learning based strategy

In this part, a reinforcement learning based strategy is designed for the IO size between 4KB and 4MB. The strategy aims to learn an optimal policy through interaction with the environment, maximizing cumulative rewards. The RL method involves two parts, agent (strategy) and environment (storage system).

In this paper, the agent observes the current state in the environment and taking an action in response of the IO request. Through repeated interactions, it refines its policy to maximize long-term rewards. RL methods mainly include policy iteration and value updates, like Q-learning and policy gradients, helping the agent eventually reach an optimal policy for effective decision-making. We formulate the RL-based strategy with Reward, State and Action designs.

1) Reward: We define the reward of each interactions as the function of the response latency for the IO request. The detail definition for the reward is shown as follows.

$$R = \begin{cases} 1/L_r & \text{no migration between SSDs} \\ max\{0, 1/L_r - R_p\} & \text{data migration appears} \end{cases}$$

where L_r, R_p and R represent the last served request latency, migration penalty and interaction reward, respectively. The definition of migration can be found in part of Action design. We select R_p to be equal to 0.001 L_e or 0.001 L_m (L_e is the time spent in evicting pages from the fast storage to the slow storage, and L_m is the time spent in move pages from the slow storage to the fast storage).

2) State: At each time-step, we collect the state features for a particular read/write request with an observation vector. We perform feature selection to determine the best state features with the concluded two principles in section A. As such, the observation vector is composed of request size, and they are listed in problem formulation of Section III B.

Algorithm 1 RL-based data placement algorithm

Input: State space $state = s_1, s_2, r_1, w_1, r_2, w_2, r'_1, w'_1, r'_2, w'_2,$
action space $action = R_{slc}, R_{QLC}, M_{eviction}, M_{migration}$
online-workload $\mathbf{P} = \mathcal{C}, a, \mathcal{A}, \mathcal{B}$
offline-patterns $P_{offline} = \mathcal{C}, a, \mathcal{A}, \mathcal{B}$
Output: IO response action R_{slc} or R_{QLC}, data placement strategy $M_{eviction}$ or $M_{migration}$
Extract the online workload patterns **P**
Compare **P** with the offline-patterns $P_{offline}$
for IO count $\in \mathcal{C}$ **do**
    Process the IO request with the write or read manager
    if IO is READ **then**
        Process with the read manager
        IO is responded by the data owner
    elseIO is WRITE
        Process with the write manager
        Case1: $IO_{size} < T^1_{write}$
        SLC SSD responds
        Case2: $IO_{size} > T^2_{write}$
        QLC SSD responds
        Case3: $T^1_{write} \geq IO_{size} \leq T^2_{write}$
        Fix the solution of $min(\mathcal{L})$
    end if
    Predict the next IO with learned features
end for
Update the State space *state*
return IO response action, data placement strategy.

3) Action: In the HSS, data can be moved between QLC and SLC SSDs, and the procedures are eviction and migration. When the space on the SLC SSD is insufficient, infrequently used data should be evicted to the QLC SSD. Conversely, if migrating data from the QLC SSD to the SLC SSD can improve latency performance, it should be moved accordingly. In a HSS, two possible actions are: placing data in the fast device or the slow device.

In final part, we present the RL-based data placement algorithm in Algorithm 1 .

V. Performance Evaluation

A. Experiment settings

We evaluate the proposed method with real storage system, and the storage layer is composed of KIOXIA KFL61HUL800G and self-developed QLC SSDs (16T). The HSS devices present themselves as a unified flat block device, offering the operating system a contiguous logical block address (LBA) space. To handle I/O requests efficiently, we developed a lightweight custom block driver interface that orchestrates operations with the underlying storage devices. Table II outlines the system specifications, highlighting the key characteristics of the two storage devices utilized in setup.

B. Real world workloads

We utilize five distinct block-I/O traces from the MSRC benchmark suite, collected from real-world enterprise environments. Additionally, we carefully select five traces exhibiting varied I/O-access patterns, as summarized below, to analyze various workloads characterized by differing IO patterns, levels of randomness and data hotness. This approach ensures comprehensive insights into varying workload behaviors.

TABLE II
System specification

System Configuration	Intel(R) Xeon(R) Gold 6330@2.00GHz CPU*2, Ethernet Controller X710 for 10GbE SFP+*2, 32 GiB RDIMM DDR4 2666 MHz*16
Storage Devices	**Specifications**
SLC SSD (SCM)	800GB, PCIe 4.0 NVMe, SLC, R/W: 6.2/6.2 GB/s, random R/W: 1480K/ 360K IOPS
QLC SSD	15.36TB, PCIe 4.0 NVMe, QLC, R/W: 6.98/1.18 GB/s, random R/W: 1341K/ 13.2K IOPS

TABLE III
Workload features

Workloads	Write %	Read %	Avg.request size	Avg.access count	No.of unique requests
hm_1	4.7	95.3	15.2	44.5	6265
web_1	45.9	54.1	29.6	1.2	6095
wdev_2	99.9	0.1	8.0	17.7	4270
usr_0	59.6	40.4	22.8	19.7	2138
proj_2	12.4	87.6	42.4	2.9	27967

C. Baseline approaches

To demonstrate the performance of the proposed approach, We compare our method with three similar HSS data placement techniques, Sibyl [2], RNN-HSS [9] and LRU.

- **LRU**: It manages data placement by tracking access frequency. Frequently used data is retained in faster storage (SLC SSD), while less-accessed data is moved to slower, high-capacity storage (QLC SSD). This approach optimizes performance by reducing latency for active workloads.
- **RNN-HSS**: In RNN-HSS [9], Doudali *et al.* come up with a method of Kleio, which is a a hybrid memory page scheduler combining lightweight, history-based data tiering with intelligent placement decisions powered by deep neural networks.
- **Sibyl**: In Sibyl [2], Singh *et al.* propose a data placement method for HSS that dynamically distributes data across different storage tiers, using machine learning to optimize access patterns and improve performance and cost efficiency.

There are also many other similar methods, such as cold data eviction [10], history-based page selection [11]. In [2], the authors have already demonstrated that the Sibyl has better performance compared to these methods. So, we will not repeat it to do the comparison in our experiment.

D. Performance Evaluation Metrics

We assess the proposed method using two metrics: average request latency and throughput (IOPS).

1) *Average request latency*: It refers to the mean latency of all storage read/write operations in a workload.
2) *Request throughput*: measures the throughput of storage requests in terms of the number of completed I/O operations per second within the workload.

E. Experiment Results

In this part, we compare the proposed method with other three state of art works, LRU, RNN-HSS and Sibyl, and demonstrate the efficiency of our method.

TABLE V shows the average request latency of four methods. We can see that our method owns better performance than LRU and RNN-HSS with all workloads. This is because that the proposed method can extract and predict the IO patterns accurately. In addition, the process of IO dynamic classification is more efficient in the resource limited storage system.

As for sibyl, the performance of our proposed method is better than it in most scenarios except for the workload of *web*_1. The reason is that the sibyl is more effective under the workload feature of low access count. LRU adopts the traditional bidirectional linked list for the data placement which is relies on simple and static rules for tracking usage, and its performance is worse than other three methods. In contrast, RNN-HSS, Sibyl and our method can extract and predict the features and access patterns of workload, and they can dynamically optimize the IO performance with the smart data placement strategy.

TABLE IV
AVERAGE REQUEST LATENCY (µS)

Method	hm_1	web_1	wdev_2	usr_0	proj_2
LRU	121.4	83.5	72.3	78.6	110.4
RNN-HSS	98.1	76.3	69.7	68.2	103.3
Sibyl	106.9	57.4	61.7	51.3	101.4
Ours	93.6	61.7	56.4	46.9	94.5

We also summarize the throughput performance for all methods on various workloads in Fig.3. It shows that our proposed method achieves the best throughput performance.

In summary, our proposed IO patterns-aware and dynamic scheduling based data placement strategy achieves the better performance of latency and throughput than other three state-of-art methods.

VI. CONCLUSION

In this paper, we have proposed the IO patterns-aware and dynamic scheduling based data placement strategy for HSS. Specifically, we first analyze the IO patterns and the different specifications for SLC and QLC SSDs. In order to alleviate the complexity of RF-based method, a threshold based IO scheduling method is designed. Finally, we have proposed a reinforcement learning based IO patterns extraction and prediction to analyze the online workloads, and optimize the data placement strategy. Experiments conducted on real storage system and workloads demonstrate that our IO patterns-aware and dynamic scheduling based data placement strategy can reduce the global system access latency while achieves better throughput.

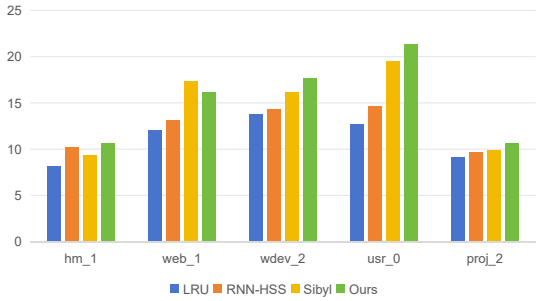

Fig. 3. Request throughput (*kIOPS*)

ACKNOWLEDGMENT

This work is partially supported by Shandong Natural Science Foundation (NO.ZR2024QF154).

REFERENCES

[1] Y. Song, W.-H. Kim, S. K. Monga, C. Min, and Y. I. Eom, "Prism: Optimizing key-value store for modern heterogeneous storage devices," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 588–602.

[2] G. Singh, R. Nadig, J. Park, R. Bera, N. Hajinazar, D. Novo, J. Gómez-Luna, S. Stuijk, H. Corporaal, and O. Mutlu, "Sibyl: Adaptive and extensible data placement in hybrid storage systems using online reinforcement learning," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 320–336.

[3] K. Wu, Z. Guo, G. Hu, K. Tu, R. Alagappan, R. Sen, K. Park, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "The storage hierarchy is not a hierarchy: Optimizing caching on modern storage devices with orthus," in *19th USENIX Conference on File and Storage Technologies (FAST 21)*, 2021, pp. 307–323.

[4] A. Raina, J. Lu, A. Cidon, and M. J. Freedman, "Efficient compactions between storage tiers with prismdb," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023, pp. 179–193.

[5] J. Ren, X. Chen, Y. Tan, D. Liu, M. Duan, L. Liang, and L. Qiao, "Archivist: A machine learning assisted data placement mechanism for hybrid storage systems," in *2019 IEEE 37th International Conference on Computer Design (ICCD)*. IEEE, 2019, pp. 676–679.

[6] R. Stoica, R. Pletka, N. Ioannou, N. Papandreou, S. Tomic, and H. Pozidis, "Understanding the design trade-offs of hybrid flash controllers," in *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2019, pp. 152–164.

[7] L. Luo, D. Yu, Y. Lv, and L. Shi, "Critical data backup with hybrid flash-based consumer devices," *ACM Transactions on Architecture and Code Optimization*, vol. 21, no. 1, pp. 1–23, 2023.

[8] C. Wang, D. Wang, Y. Chai, C. Wang, and D. Sun, "Larger cheaper but faster: Ssd-smr hybrid storage boosted by a new smr-oriented cache framework," in *Proc. IEEE Symp. Mass Storage Syst. Technol.(MSST)*, 2017.

[9] T. D. Doudali, S. Blagodurov, A. Vishnu, S. Gurumurthi, and A. Gavrilovska, "Kleio: A hybrid memory page scheduler with machine intelligence," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, 2019, pp. 37–48.

[10] C. Matsui, C. Sun, and K. Takeuchi, "Design of hybrid ssds with storage class memory and nand flash memory," *Proceedings of the IEEE*, vol. 105, no. 9, pp. 1812–1821, 2017.

[11] M. R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G. H. Loh, "Heterogeneous memory architectures: A hw/sw approach for mixing die-stacked and off-package memories," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 126–136.