2023 60th ACM/IEEE Design Automation Conference (DAC) | 979-8-3503-2348-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/DAC56929.2023.10247988

RLAlloc: A Deep Reinforcement Learning-Assisted Resource Allocation Framework for Enhanced Both I/O Throughput and QoS Performance of Multi-Streamed SSDs

Mengquan Li[1#*], Chao Wu[2#*], Congming Gao[3], Cheng Ji[4], Kenli Li[1]
[1]School of Computer Science and Electronic Engineering, Hunan University, CN
[2]Department of Electrical and Computer Engineering, Northeastern University, USA
[3]School of Information, Xiameng University, CN
[4]School of Computer Science and Engineering, Nanjing University of Technology, CN

***Abstract*—Multi-streamed Solid-State Disks (SSDs) have attracted increasing adoption in modern flash storage devices. Despite their excellent promise, effective flash resource allocation is still limiting both their achievable I/O performance and practical implementation. To this end, we develop the first-of-its-kind framework dubbed RLAlloc, which for the first time demonstrates deep Reinforcement Learning-assisted resource Allocation for boosting both I/O throughput and QoS performance of multi-streamed SSDs. Extensive experiments consistently validate the effectiveness of RLAlloc, improving up to 39.9% on I/O throughput and 44.0% on QoS performance over the state-of-the-art competitors.**

I. Introduction

Multi-streamed Solid-State Disks (SSDs) have been increasingly adopted by modern flash devices.In these devices, the data with a similar lifetime are clustered into a stream and accommodated in the same physical block of SSD devices for reducing GC overhead. In this way, the overall SSD performance is greatly improved. To accurately identify I/O data's lifetime and classify them into I/O streams, numerous advanced techniques have been proposed [1], [2].

However, effective flash resource allocation is a critical obstacle hindering the practical implementation of multi-streamed SSDs. Currently, there exist two types of resource allocation approaches on multi-streamed SSDs, i.e., static and dynamic approaches. The static approaches [3], [4] possess simplicity and easy-to-implement features but can not adapt to the real-time demands of various I/O streams [5]. In contrast, the dynamic approaches allocate flash resources to the streams according to their real-time demands [5], [6]. However, these works exacerbate I/O conflicts, incurring prolonged I/O latency and large QoS degradation.From the experimental results in Sec. IV, the state-of-the-art (SOTA) dynamic approach improves I/O throughput by 26.1% compared to the static counterparts, but incurs a largely prolonged worst-case latency of 37.2 - 67.3% at the 99.9^{th} percentile. To sum up, the existing approaches would decrease either I/O throughput or QoS performance. Therefore, it is urgent to design an effective resource allocation approach that could achieve the best of both worlds.

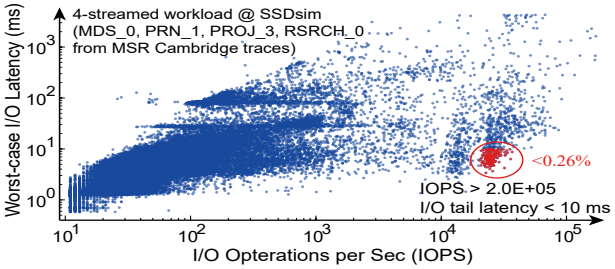

Fig. 1: The I/O throughput and QoS performance of different resource allocation schemes. There are 6.15E+04 randomly sampled allocation schemes in this figure. The design with IOPS >2.0E+05 and the worst-case I/O latency (characterizing QoS performance) <10 ms are marked as red, which are sparse (>0.26%) within this design space.

Recently, the significant progress of deep reinforcement learning (RL) has made it widely used to solve high-dimensional tasks [7]. Nevertheless, it is challenging to solve the flash memory resource allocation problem with RL. The reason is three-fold. First, it needs expert knowledge of both deep RL techniques and SSD devices, as well as the ability to properly integrate them from the operating system (OS) level down to the device level. Second, flash resource allocation is complicated with a large design space composing of diverse I/O stream patterns, different running statuses of SSDs, various action choices for resource allocation. Finally, as shown in Fig. 1, satisfactory allocation candidates are extremely sparse in the large space. A majority of allocation schemes would cause low I/O throughput and high tail latency, which further exacerbates the difficulty for effective SSD resource allocation.

To tackle the aforementioned bottleneck, we develop the first-of-its-kind framework dubbed RLAlloc, which for the first time demonstrates deep Reinforcement Learning-assisted resource Allocation for boosting both I/O throughput and QoS performance of multi-streamed SSDs. Specifically, our RLAlloc consists of two main contributions: (1) a cross-domain resource allocation approach, which perfectly incor-

[#]The authors contribute equally.
[*]Corresponding authors. Email: chaowu916@gmail.com.

This work is partially supported by the National Key Research and Development Program of China (No.2021YFB2206603), NSFC (62202159 and 62106146) and NTU NAP (M4082282), Singapore.

porates deep reinforcement learning in modern SSDs' resource allocation procedure with excellent effectiveness but marginal overheads; and (2) a comprehensive design space definition, which considers the most-ever influential factors in flash resource allocation (including the state factors related to I/O stream patterns and SSD execution statuses, the action choices of resource allocation, and the dual-objective based reward (i.e., I/O throughput and QoS) when taking actions in these states), so as to maximize both the I/O and the QoS performances of multi-streamed SSDs. Extensive experiments consistently validate the excellent effectiveness of the RLAlloc framework, improving average I/O throughput by 25.1% and reducing the worst-case I/O tail latency at the 99.9^{th} percentile by 9.2–44.0% over the state-of-the-art competitors.

II. Background and Prior Art

Multi-streamed SSD. Multi-streamed SSDs have been acknowledged thanks to their excellent storage performance and low GC overhead [8]. Fig. 2 illustrates the architecture of multi-streamed SSDs. An SSD device communicates with the host using multiple channels. Each channel connects with a set of chips, each chip accommodates several dies, and each die contains several planes. Several hundreds of blocks are contained in each plane, and each block consists of multiple pages. A device controller is used for SSD resource allocation. Multi-streamed SSD technology isolates I/O data with different lifetimes to disparate SSD blocks and clusters the data with similar hotness to a block, thus reducing GC overhead and improving overall SSD performance. For example, Block 0 accommodates Stream 0 constituted by RQ0 - RQ2 and Block 1 accommodates Stream 1 constituted by RQ3 - RQ5.

Deep RL. Deep RL has attracted great attention in solving high-dimensional problems [9]. There are four essentials in RL models, i.e., agent, state, action, and reward. The agent learns from interacting with an environment with trial and error. Due to the explosive increase of state and action spaces, instead of storing the q-value in a tabular form, deep RL adopts a deep-Q-network (DQN) for value function approximation and also to learn useful representations of high-dimensional raw inputs, achieving better generalization and cost-saving. The DQN would be pre-trained using an offline training dataset and continue performing online learning during inference. The training dataset is a set of 4-tuples (old_state, $action$, $reward$, $next\_state$) collected beforehand while exploring the simulation. Randomly-sampled mini-batches are used to train the DQN on iterations until the model is converged. After being well trained, on each step, the agent acquires the current state by perceiving from the environment, and then selects the optimized action according to the inference results of the DQN (i.e., the Q-values for all possible actions). After the selected action has been taken, the agent gets a reward from the environment. The reward is then updated and also used in the DQN for online learning. Finally, the environment is pushed into the next state.

Flash resource allocation approaches. The current flash resource allocation approaches can be classified into two types:

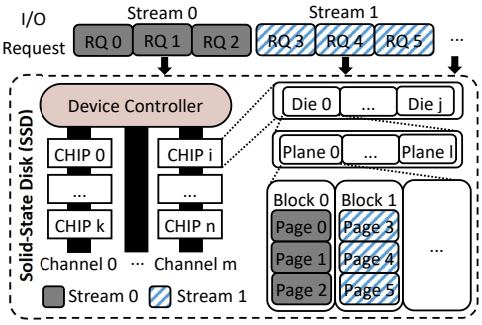

Fig. 2: The multi-streamed SSD.

static approaches [3], [4], [10]–[12] and dynamic approaches [5], [6]. Thanks to the simplicity and the easy-to-implement feature, static approaches have been prominent. However, as they allocate flash resources for each stream during initialization, they cannot meet the real-time demands of various I/O streams. Besides, it is hard to precisely predict the demand of I/O streams beforehand and allocate resources accordingly. Imbalanced resource allocation is incurred [5], which causes either resource waste or excessive GC activities, significantly degrading I/O throughput. Dynamic allocation approaches have been proposed achieving excellent I/O throughput. The SOTA dynamic approach [5] allocates SSD resources according to the real-time demand of each I/O stream at the unit of flash chip. However, it breaks the performance isolation among I/O streams. Multiple I/O streams potentially access the same flash chip, incurring a large amount of I/O conflicts. Besides, this work is also unable to reduce GC, as it fails to comprehensively consider all the influential factors during SSD resource allocation, e.g., the time and the space utilization of flash resources. If taking all the factors into account, a large search space is created, which is beyond the capability of conventional approaches. By leveraging the powerful learning and decision-making abilities of deep RL, we propose RLAlloc, an AI-assisted SSD resource allocation framework.

III. The Proposed RLAlloc Framework

Fig. 3 illustrates an overview of the proposed RLAlloc framework. Multiple I/O requests (e.g., RQ_0, RQ_1, ... RQ_n) are produced from the host side, waiting to be allocated. They are first clustered into multiple I/O streams via the techniques proposed in [1], [2], [13], [14]. Then, RLAlloc allocates appropriate flash resources to accommodate these streams, achieving maximized I/O throughput and QoS performance with marginal overheads. The RLAlloc can be integrated into the device controller, functioning as a resource allocator.

Our RLAlloc contains 4 steps: ① State Observation. First, RLAlloc identifies the current state $\vec{S}$ by collecting all of the state factors from both the host and the device sides, including those related to I/O stream patterns and SSD device running statuses. Formally, $\vec{S} = (s_0, s_1, ..., s_m)$ where m is the total number of state factors. ② Action Selection. All possible actions would also be defined in advance, termed as action candidates. The state and action spaces will be elaborated in Sec. III-A. Besides, RLAlloc will choose to per-

form exploitation or exploration using ε-Greedy policy, similar in [15]. When performing exploration, RLAlloc randomly selects one action after DQN inference. While if performing exploitation, RLAlloc will choose the action corresponding to the maximum q-value based on the DQN inference results. An action selection policy is determined via step ②, which will be used to select the desired action after finishing DQN inference. ③ DQN Inference. Based on the current state $\vec{S}$ and the action candidates, RLAlloc calculates the Q-values for all possible actions through inference (e.g., q_0, q_1, q_t, where t action alternatives in total). Then, an action (e.g., A^*) will be selected according to the determined policy from step ②. The replay buffer enables applying the experience replay mechanism, which facilitates fast training, similar in [16]. ④ On-line learning. Next, the SSD device takes the action A^* and reports the actual reward R^* to the DQN. Set R^* as the ground truth value, DQN updates weights using back propagation (BP). In this way, online learning is realized.

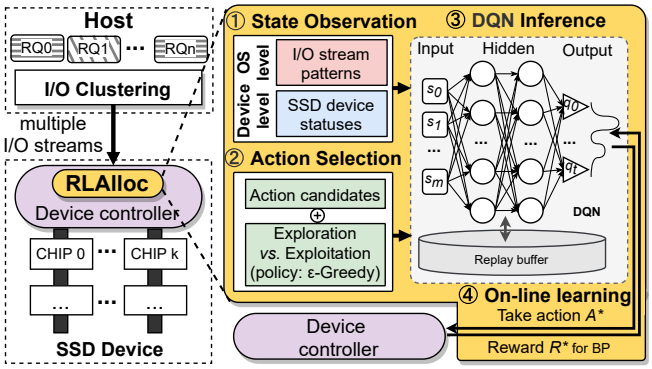

Fig. 3: The proposed RLAlloc framework.

A. Design space definition

A comprehensive design space is a prerequisite of our RLAlloc framework for decision-making exploration and optimization, which we summarize in Fig. 4.

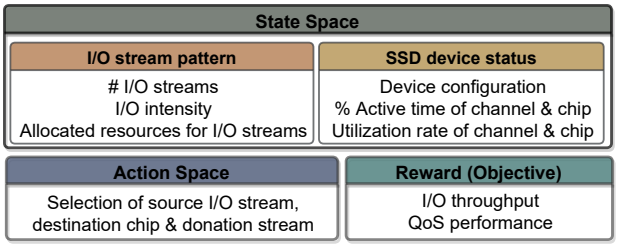

Fig. 4: Design space of the proposed RLAlloc framework.

*1) **State Space**:* The state space is constituted by the factors related to I/O stream patterns and the real-time running statuses of SSD devices. The factors are explained below.

- *# I/O stream:* # of I/O streams from the host side.
- *I/O intensity of each stream.* This factor characterizes the I/O demand (i.e., the number and the size of read I/Os and write I/Os in the streams). Given I/O requests from the host side, SSD devices will split each of them into one (or multiple) transactions which are read/written as fixed-size pages. We define the I/O intensity of each stream as a 2-tuple, i.e., $(\#Trans_{rd}, \#Trans_{wt})$, which respectively counts the number of the read and write transactions during the designated period of time.
- *Allocated resources for each I/O stream.* Our RLAlloc considers the hardware flash resources at both the channel and the chip levels, as [5] do. This factor shows the channels and chips that are already allocated to each I/O stream. A recording list is maintained for each stream.
- *Device configuration.* This factor is an abstraction to a physical SSD device, by recording the number and the size of flash resources (such as channels, chips, dies, planes, blocks and pages) in the SSD.
- *Active time percentage of each channel and chip.* This factor refers to the percentage of active time of each channel and chip over the designated period of time, reflecting the time utilization of the SSD resources.
- *Utilization rate of each channel and chip.* This factor shows the space utilization of each channel and chip, i.e., the ratio of occupied space to the total space. The last two factors together consider both the time and the space utilization of channel bandwidth and storage resources, which enable our RLAlloc to avoid imbalanced resource allocation and the in-device activation of GC.

*2) **Action Space**:* A resource allocation action consists of three sub-actions. The first two are the selections of a source flash chip (which has free space and is available to be allocated to accommodate I/O data) and a destination I/O streams (which are requesting flash resources). Then we can assign the source chip to the destination stream. While if the source chip has been allocated to another I/O stream, termed the source stream, it is the third sub-action to select the source stream. In such cases, the source stream would spare part of flash chips for the destination stream and the two streams will share the source chip. For the resource allocation problem on multi-streamed SSDs, there are multiple options for each sub-actions as there are multiple I/O streams and numerous SSD flash chips. It constitutes a large action space when combining them together, in which different combinations (i.e., the points in the action space) will lead to distinct storage performance.

*3) **Reward**:* RLAlloc aims to optimize both I/O throughput and QoS performance. We adopt a metric, the I/O tail latency (i.e., the worst-case I/O latency), to describe the QoS performance. This metric characterizes the devices' ability to provide stable I/O performance. Therefore, the reward can be formulated as follow:

$$Re = W \cdot \vec{Obj} \qquad \vec{Obj} = [IOPS,\ L_{IO}^{tail}] \tag{1}$$

where W is a weight vector, which is a hyper-parameter that reflects the relevant importance of each performance metric. $\vec{Obj}$ is an objective vector consisted by two metrics (i.e., $IOPS$ and L_{IO}^{tail}). $IOPS$ denotes the I/O throughput (i.e., the number of completed I/O requests during the designated period of time), which is a positive objective that RLAlloc strives to maximize. L_{IO}^{tail} is the worst-case I/O latency, which is a negative objective that RLAlloc strives to minimize.

B. DQN model

A Multi-layer Perceptron (MLP)-based DQN is adopted in RLAlloc as it is capable of modeling high-dimensional functions and features a mature training infrastructure [17]. The DQN contains an input layer, an output layer and multiple hidden layers. We now provide details for the DQN model.

*1) **Input and Output Vectors.**:* We elaborate on the input vector (i.e., state vector $\vec{S}$) representation below.

1) *# I/O stream:* a scalar showing # of multiple streams.
2) *Stream pattern:* a tuple representing the # reads, # writes and a list recording the allocated chips for each stream. For example, for a SSD with 8 chips, this tuple contains at most 10 elements (i.e., 1+1+8) for each stream.
3) *Device configuration:* a 7-tuple indicating the number of channels, chips, dies, planes, blocks, pages and the size of pages in a SSD device.
4) *Channel utilization:* a scalar recording the active time percentage for each channel.
5) *Chip utilization:* a 2-tuple recording the time and the space utilization rates for each chip.

Given I/O demands and a SSD device, the maximum size of the input vector can be calculated as $1 + N_{str} \cdot |SP| + 7 + N_{cn} + 2N_{cp}$, where N_{str}, N_{cn}, and N_{cp} are respectively the # of I/O streams, channels and chips. $|SP|$ is the length of the tuple *Stream pattern*.

After inference, the Q-values of all possible actions will be outputted. Therefore, the output vector has N_{act} neurons, where N_{act} is the number of action candidates.

*2) **Topology and Training.**:* In this work, we choose a 4-layer deep MLP. Based on the SSD configuration shown in Tab. II, the DQN contains 224 input neurons, 57 output neurons and 2 hidden layers with 500 neurons per layer.

We train the model with 10G samples collected by SSDsim during the design time, which is a set of pairs constituted by input vectors and the desired action (functioning as labels). This model is pre-trained offline and keeps online learning while inference by using the Stochastic Gradient Descent (SGD) optimizer with a learning rate of 10^{-2}, an exploration rate of 10^0 which is decayed to 10^{-2} among 100K epochs, and a batch size of 1000. The loss function for both offline training and online learning complies with that in [9].

C. Design overhead

The overhead of RLAlloc is mainly incurred by 1) the cost of collecting state information from both host and device side, and 2) the storage and inference costs of the DQN.

State collection. We can adopt open-channel SSD technology [18] for realizing state observation, which proposes to expose the SSD internals and enable a host to control the data placement and physical I/O scheduling. In this way, the state information related to both the SSD device statuses and the I/O stream patterns can be easily collected. Moreover, with the progress of powerful SSD controller and the advance in deep RL, RLAlloc can also be integrated in the device controller in the near future. Then all the state information can be readily fetched from SSD devices. We leave optimizing this problem an important future work.

DQN model size. When weights are represented as 32-bit floats, the DQN takes 1.5MB of storage. With advanced pruning and quantization techniques [16], [19], it can be compressed (<200KB) with marginal storage cost.

DQN inference time. We test the inference time of our DQN by running it on a workstation with Intel(R) CPU i5 at 3.50 GHz and 32 GB memory. The results show that each inference operation averagely takes 15.3 μs. As our DQN keeps online learning while inference, it consumes additional time for BP training, which takes 21.7 μs on average with the help of advanced pruning technique. To sum up, it takes <40 μs for DQN inference, achieving a high frequency of 2.5×10^4 inference per second.

IV. Performance evaluation

Workload. We studied our RLAlloc framework on a modified MSR Cambridge I/O workload [20], which is a well-recognized online repository. In this workload, sets of I/O traces are collected from the servers using SSD devices [21]. Each set of I/O traces is capsuled as a single I/O stream. We mix multiple sets of I/O traces to mimic the real-world multi-streamed I/O workload. Tab. I shows the characteristics of these I/O streams. They show diversities in I/O intensity (including the total I/O volume and the numbers of I/O requests), read and write I/O ratio and the average arrival time interval of adjacent I/O requests.

TABLE I: The characteristics of the I/O workload [21].

Name	Volume (GB)	# I/O	% Write	Interval (ms)
MDS_0	7.72	1.21E+06	88.11%	0.4962
PRN_1	2.27	1.12E+06	24.66%	0.535
PROJ_3	2.75	2.24E+06	5.18%	0.2691
RSRCH_0	11.34	1.43E+06	90.68%	0.4217
SRC1_1	1.82	4.57E+06	4.74%	0.126
STG_0	15.82	2.03E+06	84.81%	0.2976
TS_0	11.89	1.80E+06	82.42%	0.3850
WEB_2	0.82	5.18E+06	0.75%	0.116

Methodology. We compared the proposed RLAlloc framework against 2 SOTA competitors, including a static flash resource allocation approach [12] and a dynamic one [5], dubbed as *Static* and *Dynamic* respectively.

TABLE II: SSDsim Configuration.

Parameter	Value	Parameter	value
Capacity	128GB	Channel number	8
Chip per channel	8	Die per chip	2
Plane per die	2	Block per plane	2048
Page per block	64	Page capacity	4KB
Over provisioning ratio	10%	Block erase latency	5 ms
Page read latency	75 μs	Page write latency	1100 μs
Garbage collection scheme	Greedy	Page Allocation	Dynamic

We build RLAlloc in C to facilitate its practical implementation on SSD devices. As a proof of concept, we adopt a trace-driven simulator, SSDsim [22], to mimic the real-time status of multi-streamed SSDs. SSDsim is an open-source SSD simulator that is widely used by the storage community for design and verification, thanks to its high simulation accuracy and advanced-command support [15]. Tab. II shows the SSDsim configuration. The latencies of page reading, writing

and erasing are obtained based on commercial Intel SSD products [23]. We integrated the various resource allocation approaches, including our RLAlloc, *Static* and *Dynamic* into SSDsim for fast feasibility and effectiveness validation. We leave the practical implementation of RLAlloc on actual SSD devices as future work.

I/O Throughput. Fig. 5 illustrates the I/O throughput of different flash resource allocation approaches on various I/O traces. RLAlloc outperforms *Static* by 25.1% on average, with a minimum of 15.4% and a maximum of 39.9%. In addition, compared with *Dynamic* which achieves the SOTA I/O throughput, RLAlloc has only a slight difference (i.e., 0.8% on average) but has greatly enhanced QoS performance as testified later.

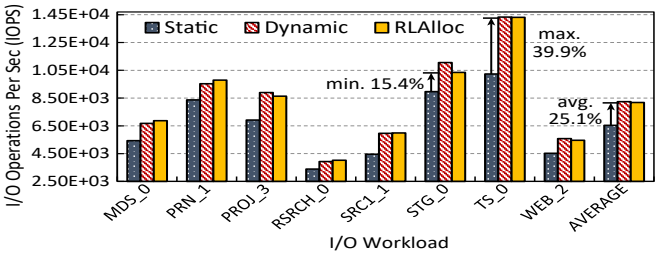

Fig. 5: I/O throughput comparison of our RLAlloc with the SOTA flash allocation approaches on various I/O traces.

Thanks to the adaptive allocation capability, RLAlloc and *Dynamic* achieve higher I/O throughput than *Static*. We can elaborate on such results from the perspectives of both I/O streams and SSD devices. From the view of I/O streams, *Static* approach cannot cope with various I/O demands and thus easily results in imbalanced resource allocation. In contrast, RLAlloc and *Dynamic* allocate flash resources adapting to the real-time demand of each stream. The streams are served well thanks to sufficient SSD resources. From the view of SSD devices, RLAlloc and *Dynamic* could maximize the internal parallelism of SSD devices. While *Static* is probably to expose the parallel processing unit of SSDs into long idle time because of imbalanced allocation. We summarized and compared the active percentages of flash chips when adopting different approaches in Fig. 6. Although based on a sparse I/O pattern, *Static* still shows the lowest chip utilization rate. Both RLAlloc and *Dynamic* achieve higher parallelism. Compared with *Dynamic* who shows volatile storage performance, our RLAlloc's performance fluctuates at the very beginning but soon stabilizes and remains stable, which further validates the robustness of RLAlloc.

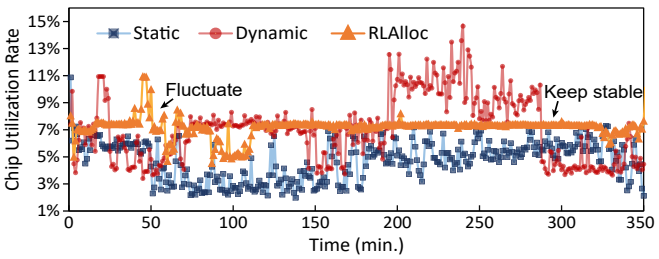

Fig. 6: The active percentage of flash chips when adopting different allocation approaches.

QoS Performance. Fig. 7 illustrates the cumulative distribution of tail 1% I/O latency of different approaches on various I/O traces, from which we make two observations. First, *Static* achieves the best QoS performance over *Dynamic* and RLAlloc. That is because when using *Static*, SSDs will provide dedicated flash resources for each stream and the stream possesses the full bandwidth of channels and all the storage capability of flash chips. Second, compared with *Dynamic*, our RLAlloc achieves a big improvement on QoS Performance, reducing the tail latency by 16.1–35.0% at the 99^{th} percentile and by 9.2–44.0% at the 99.9^{th} percentile.

Fig. 8 is presented to explain the above results. I/O conflict is notoriously one main culprit of QoS performance degradation [15]. *Static* decreases the number of I/O conflicts by isolating I/O streams. While *Dynamic* fails to do that as multiple I/O streams might share the same flash chip, exacerbating I/O contention. As shown in Fig. 8(a) that the number of I/O conflicts when adopting *Dynamic* increases by 60.9% over that adapting *Static*. By comprehensively taking the I/O intensity and the device active status into account, RLAlloc reduces I/O conflicts by 33.4% on average over *Dynamic*. In addition, GC is another reason that leads to prolonged I/O tail latency. *Static* could incur excessive GC activities when insufficient flash resources are allocated. *Dynamic* fails to mitigate this issue as it does not consider the space utilization of each flash chip, while which matters for GC optimization. In contrast, RLAlloc significantly avoids the excessive activation of GC by allocating chips with sufficient free space to I/O streams with high resource demand. As shown in Fig. 8(b), RLAlloc reduces GC activities by 59.2% over *Static*.

In summary, the existing approaches cannot reach both excellent I/O throughput and favorable QoS performance, as they cannot reach the balance between performance isolation and high device parallelism utilization. By combining all influential factors into consideration in resource allocation, RLAlloc enhances SSDs' internal parallelism, and meanwhile, reduces the numbers of I/O conflict and GC activation. In this manner, RLAlloc enhances both the I/O throughput and QoS performance for multi-streamed SSDs.

Sensitivity study. The efficiencies of resource allocation approaches are sensitive to the number of I/O streams. To evaluate the optimization efficacy of RLAlloc under different numbers of I/O streams, we performed and compared RLAlloc and the SOTA approaches based on STG_0 traces. To save space, we presented the results of I/O throughput and I/O tail latency at the 99.9^{th} percentile, as illustrated in Fig. 9.

Three observations could be obtained from the results. First, the fewer I/O streams are, the fewer performance differences exist among various approaches. For example, RLAlloc outperforms *Static* by only 0.9% in terms of I/O throughput when there are 2 streams. When I/O streams increases to 20, the throughput gain of RLAlloc over *Static* rises to 24.3%. A similar trend is shown in QoS performance. The reason is that fewer I/O streams incur fewer flash resource conflicts among I/O streams, thus the optimization space is limited. Second, *Dynamic* always shows a better I/O throughput but at the

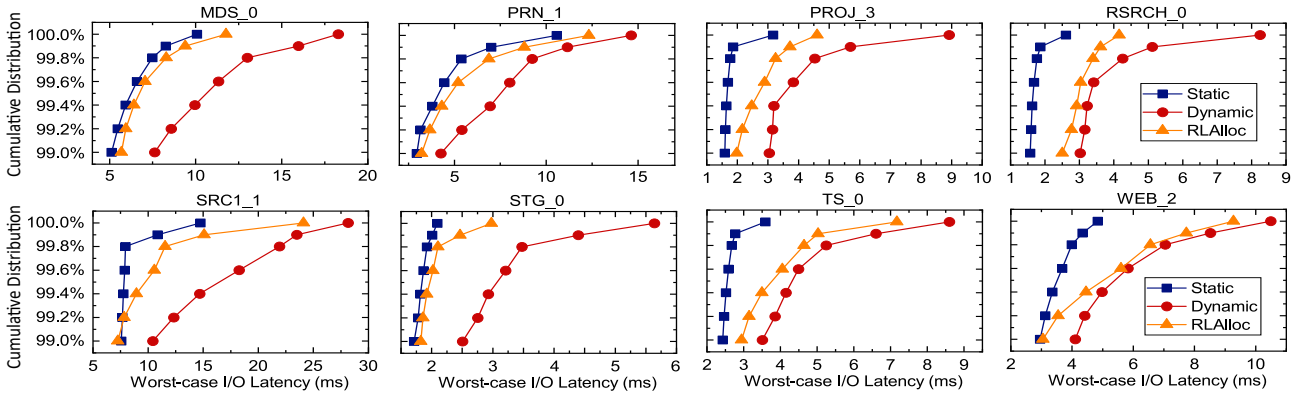

Fig. 7: The worst-case I/O latency comparison of RLAlloc with the SOTA competitors on various I/O traces.

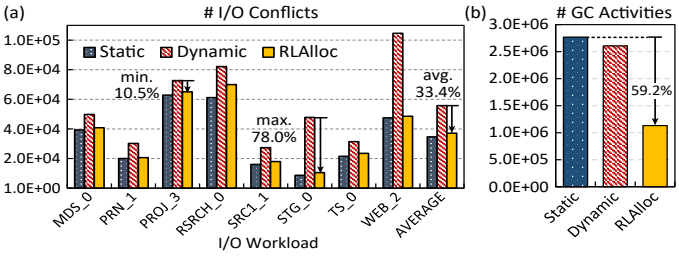

Fig. 8: (a) The number of I/O conflicts and (b) the number of GC activities when adopting different allocation approaches.

cost of degraded QoS performance over *Static*. *Static* tends to improve QoS performance through I/O stream isolation, while *Dynamic* breaks such isolation to enhance the utilization of device internal parallelism (and thus I/O throughput). Third, RLAlloc is always capable of achieving both excellent I/O throughput and QoS performance. By considering the most-ever influential factors in resource allocation, RLAlloc balances performance isolation and maximized device internal parallelism, so as to achieve the best of both worlds.

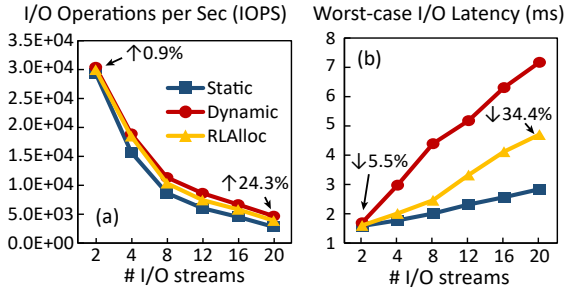

Fig. 9: The sensitivity study of I/O throughput and I/O tail latency at the 99.9^{th} percentile when adopting different allocation approaches based on the STG_0 traces.

V. Conclusion

We propose RLAlloc, the first framework to demonstrate deep reinforcement learning-assisted resource allocation for enhancing both I/O throughput and QoS performance of multi-streamed SSDs. RLAlloc perfectly incorporates deep reinforcement learning in modern SSDs' resource allocation procedure, making a great effort to pave the way for new-generation smart flash devices. Extensive experiments consistently validate the excellent effectiveness of RLAlloc, with improvements of up to 39.9% on I/O throughput and 44.0% on QoS performance over the SOTA competitors.

References

[1] T. Kim *et al.*, "Fully automatic stream management for multi-streamed ssds using program contexts," in *FAST*, 2019, pp. 295–308.

[2] E. Rho *et al.*, "Fstream: Managing flash streams in the file system," in *FAST*, 2018, pp. 257–264.

[3] D. Chang *et al.*, "Vssd: performance isolation in a solid-state drive," *ACM TODAES*, vol. 20, no. 4, pp. 1–33, 2015.

[4] J. Huang *et al.*, "Flashblox: Achieving both performance isolation and uniform lifetime for virtualized ssds," in *FAST*, 2017, pp. 375–390.

[5] B.-S. Kim, "Utilitarian performance isolation in shared ssds," in *HotStorage Workshop*, 2018.

[6] W. Choi *et al.*, "Fair resource allocation in consolidated flash systems," in *HotStorage Workshop*, 2019.

[7] P. Henderson *et al.*, "Deep reinforcement learning that matters," in *AAAI*, vol. 32, no. 1, 2018.

[8] J.-U. Kang *et al.*, "The multi-streamed solid-state drive," in *HotStorage Workshop*, 2014.

[9] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[10] M. Kwon *et al.*, "Dc-store: eliminating noisy neighbor containers using deterministic i/o performance and resource isolation," in *FAST*, 2020, pp. 183–191.

[11] J. González and M. Bjørling, "Multi-tenant i/o isolation with open-channel ssds," in *Nonvolatile Memory Workshop*, 2017.

[12] J. Yang *et al.*, "Architecting flash-based solid-state drive for high-performance i/o virtualization," *IEEE computer architecture letters*, vol. 13, no. 2, pp. 61–64, 2014.

[13] T. Kim *et al.*, "Pcstream: automatic stream allocation using program contexts," in *HotStorage Workshop*, 2018.

[14] J. Yang *et al.*, "Autostream: automatic stream management for multi-streamed ssds," in *ACM SYSTOR*, 2017, pp. 1–11.

[15] C. Wu *et al.*, "Maximizing i/o throughput and minimizing performance variation via reinforcement learning based i/o merging for ssds," *IEEE TC*, vol. 69, no. 1, pp. 72–86, 2019.

[16] ——, "Pruning deep reinforcement learning for dual user experience and storage lifetime improvement on mobile devices," *IEEE TCAD*, vol. 39, no. 11, pp. 3993–4005, 2020.

[17] K. Hegde *et al.*, "Mind mappings: enabling efficient algorithm-accelerator mapping space search," in *ASPLOS*, 2021, pp. 943–958.

[18] M. Bjørling *et al.*, "Lightnvm: The linux open-channel ssd subsystem," in *FAST*, 2017, pp. 359–374.

[19] Y. Wang, "Towards ultra-efficient dnn inference acceleration on edge devices for wellbeing applications," in *HealthDL*, 2020, pp. 17–17.

[20] D. Narayanan *et al.*, "Migrating server storage to ssds: analysis of tradeoffs," in *ACM EUROSYS*, 2009, pp. 145–158.

[21] M. Kwon *et al.*, "Tracetracker: Hardware/software co-evaluation for large-scale i/o workload reconstruction," in *IEEE IISWC*. IEEE, 2017, pp. 87–96.

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

[23] Intel, "Intel ssd 5 series." https://ark.intel.com/content/www/us.html.