Reinforcement Learning-Based SLC Cache Technique for Enhancing SSD Write Performance

Sangjin Yoo
Sungkyunkwan University, Samsung Electronics

Dongkun Shin
Sungkyunkwan University

Abstract

Although quad-level-cell (QLC) NAND flash memory can provide high density, its lower write performance and endurance compared to triple-level-cell (TLC) flash memory are critical obstacles to the proliferation of QLC flash memory. To overcome such drawbacks of QLC flash memory, hybrid architectures, which program a part of QLC blocks in the single-level-cell (SLC) mode and utilize the blocks as a cache of remaining QLC blocks, are widely adopted in the commercial solid-state disks (SSDs). However, it is challenging to optimize various parameters of hybrid SSDs such as the SLC cache size and the hot/cold separation threshold. In particular, the parameters must be adjusted dynamically by monitoring the change on I/O workloads. However, current techniques use fixed parameters determined heuristically. This paper proposes a reinforcement learning-based SLC cache management technique. By observing workload pattern and internal status of hybrid SSD, it determines the optimal SLC cache parameters maximizing the efficiency of hybrid SSD. Experimental results show that the proposed technique improves write throughput and write amplification factor by 77.6% and 20.3% on average, respectively, over the previous techniques.

1 Introduction

Recently, quad-level-cell (QLC) NAND flash memory, which can store four bits per cell, is becoming a mainstream storage medium of solid-state drives (SSDs). QLC flash memory has a higher density, and lower cost compared to single-level-cell (SLC) or triple-level-cell (TLC) flash memories. However, QLC flash memory has slower performance and lower endurance than its previous generation memories, as shown in Table 1. In order to hide the slow performance of QLC flash memory, most recent QLC-based SSDs adopt a hybrid SSD architecture which has a partitioned SLC region. The flash blocks in the SLC region are programmed in the SLC mode (i.e., they store only one bit per cell.), and thus they provide a shorter access latency than QLC blocks. The SLC region is utilized as a cache space of the remaining QLC region. By writing frequently-updated data at the SLC region, the overall performance of hybrid SSD can be improved.

In order to design a hybrid SSD, two important factors must be determined. The first is the size of the SLC region, which must be determined considering the trade-off between capacity loss and SLC-to-QLC block migration overhead. Since the capacity of an SLC block is smaller than that of a QLC block, the total capacity of SSD is reduced as more flash blocks are allocated to the SLC region. On the contrary, a too small SLC region will result in high migration costs while increasing the latency of write requests and the write amplification ratio. This is because the data in the SLC region must be migrated to the QLC region when the free blocks in the SLC region are insufficient.

The second factor is the hot/cold separation threshold. Considering the SLC-to-QLC migration cost, it is better to write only frequently-updated data (hot data) at the SLC region. Other cold data need to be sent directly to the QLC region. To discriminate between hot data and cold data, the hybrid SSD needs to observe several factors of write requests such as data size, target address, and update frequency. A widely used simple heuristic approach is to use the data size since small data tend to be frequently-updated [7, 10]. If the data size is smaller than a threshold, the data can be regarded as hot data. As a lower threshold is used, the write traffic to the SLC region decreases, and thus the SLC-to-QLC migration cost also decreases. However, more number of write requests will be sent to the QLC region, resulting in the overall write performance degradation. Therefore, the SLC cache size and the hot/cold separation threshold must be determined by considering the workload patterns and the internal behavior of hybrid SSD such as migration cost. Besides, these factors must be adjusted according to the system state change.

However, the existing techniques use heuristically-determined fixed design factors and do not adjust the factors at run time. In this paper, we propose a reinforcement learning (RL)-based SLC cache management technique for hybrid SSDs. To the best of our knowledge, our technique is the first

Table 1: Characteristics of SLC, TLC and QLC memories [12]

	SLC	TLC	QLC
Program time (page)	160 us	730 us	3102 us
Read time (page)	30 us	66 us	140 us
Erase time (block)	3 ms	4.8 ms	3.5 ms
Endurance (Max. P/E)	100,000	3,000	1,000

machine learning-based dynamic approach for hybrid SSD. Our RL-based technique defines several states that reflect the characteristics of the workload and the internal behavior of hybrid SSD. It determines the optimal SLC cache parameters using the Q-learning algorithm. Experimental results show that the RL-based technique can improve the SSD write performance compared to the existing techniques, by dynamically adjusting the SLC cache factors based on the system states.

2 BACKGROUND AND MOTIVATION

2.1 Hybrid SSD Architecture

Figure 1 shows a typical hybrid SSD architecture. As mentioned before, the flash blocks are divided into an SLC region and a QLC region. The key parameters, which have a strong influence on the write performance, are the hot/cold separation threshold (θ) and the SLC region size. The write requests with the data size not larger than θ are sent to the SLC region, whereas other requests are sent to the QLC region. Some old data in the SLC region can be migrated into the QLC region when the free space in the SLC region is insufficient.

According to the SLC region management scheme, there are two types of hybrid SSDs: static scheme and dynamic scheme. While the static scheme maintains a fixed size of SLC region and a fixed hot/cold threshold [4, 7, 8], the dynamic scheme can adjust the SLC region parameters depending on the system states such as amount of stored data, I/O access pattern, and garbage collection cost [6, 13].

Recently, several QLC SSD products began to adopt the dynamic scheme-based hybrid SSD architecture [1–3]. They use a simple technique to adjust the SLC cache size. A proper SLC region size for a given storage utilization is investigated at offline with representative workloads. At run time, the SLC region size is adjusted by referring to the offline result as the storage utilization changes. Therefore, their performance can be degraded under unexamined or variable workloads.

2.2 Motivation

To observe the problem of the current dynamic hybrid SSDs, we performed experiments with our own trace-driven hybrid SSD simulator. Two real-world workloads were used: PC and YCSB-A. The PC workload trace was collected at Windows

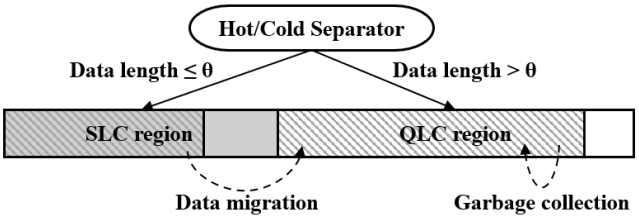


Figure 1: Typical hybrid SSD architecture.

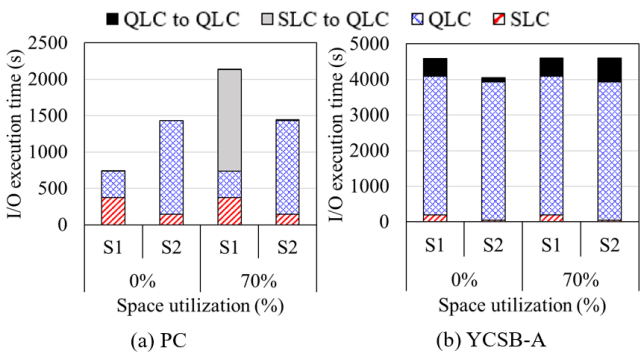
Table 2: Two example settings on SLC cache size at different storage utilizations (%)

Space utilization (%)	0 ∼20	20 ∼30	30 ∼40	40 ∼50	50 ∼60	60 ∼70	70 ∼100
Setting 1	56	50	40	30	25	20	10
Setting 2	40	40	30	25	20	10	5

10-based desktop, and it includes a larger number frequently-updated data compared to the YCSB-A database workload [5]. The total capacity of the SSD was set to 32 GB. We examined the hybrid SSD performance with two different SLC cache setting policies, as shown in Table 2, each of which uses different SLC cache sizes at different storage utilizations. The data of Setting 1 are originated from a commercial SSD [3]. Setting 1 (S1) utilizes the SLC cache more aggressively than Setting 2 (S2). The hot/cold separation thresholds (θ) of Setting 1 and Setting 2 are 64KB and 16KB, respectively.

Figure 2 compares the total I/O execution times under different policies and different space utilization values. The execution time is divided into four parts based on write type: host writes to SLC cache (SLC), host writes to QLC region (QLC), SLC-to-QLC migrations (SLC-to-QLC), and garbage collections within QLC region (QLC-to-QLC). From the results, we can know that the better policy between Setting 1 and Setting 2 is different depending on the workload and the storage utilization. For example, in the PC workload, when the initial storage is empty (i.e., the utilization is 0%), Setting 1 shows a better result than Setting 2 since many write requests are serviced at the SLC region without significant SLC-to-QLC migration overhead. However, when the utilization is high, Setting 1 shows a worse result due to high migration cost. In the YCSB-A workload, Setting 1 shows a worse result at low utilization. Since Setting 1 allocates more flash blocks to the SLC region, the capacity of the QLC region is small. Therefore, the QLC garbage collection cost increases.

Consequently, we need a more intelligent algorithm to adjust the SLC cache configuration considering the dynamically changing system states. The traditional adaptive control algorithms could be a solution. However, it is hard to design and tune an algorithm which can effectively handle the various system states of hybrid SSD under dynamically changing user workloads. We adopt a reinforcement learning (RL)-based method. Without any prior knowledge about user workload or storage characteristics, the RL-based approach can learn



(a) PC (b) YCSB-A

Figure 2: Write performance under different SLC cache management policies.

how to find the optimal management policy. Therefore, it can be used for any user scenarios and any storage products.

2.3 Reinforcement Learning and its Applicability to Dynamic SLC Cache

In reinforcement learning, an agent learns to act optimally through observations and rewards from the environment. The purpose of reinforcement learning is to choose the action that maximizes cumulative reward. We employ Q-learning, which is a widely used reinforcement technique [11]. In Q-learning, the agent calculates Q-values that tell the agent which action is right in a given state. The agent chooses an action a in any given state s, and observes the reward r and the next state s'. Then, the Q-value is updated as follows:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_a Q(s',a') - Q(s,a)) \quad (1)$$

where α and γ are the learning rate and the discount factor, respectively, a' is the action in the next state s'. The Q-learning maintains a key data structure, called Q-table, to store $Q(s,a)$. The size (# of entries) of Q-table is # of states $\times$ # of actions. We employ the ε-greedy algorithm for exploration that can maximize the total reward in the long time interval. In Equation (2), ε is a probability between 0 and 1. The agent mostly selects the action a^* in a given state s by exploiting the learned policy. At a low probability of ε, the agent selects a random action to explore the optimal policy of the changing environment. We set ε to 0.07 in our experiments.

$$\pi(s) = \begin{cases} a^* = \arg\max_a Q(s,a), & 1-\varepsilon \\ a \neq a^*, & \varepsilon \end{cases} \quad (2)$$

Figure 3 shows how the reinforcement learning can be applied to the dynamic SLC cache. At each time step, the agent, i.e., the SLC cache manager, selects an action A_t including the changes of the SLC cache size and the hot/cold separation threshold. The environment defines the state S_t based on the workload characteristics and the internal status of the SSD, and estimates the reward R_t for the previous action.

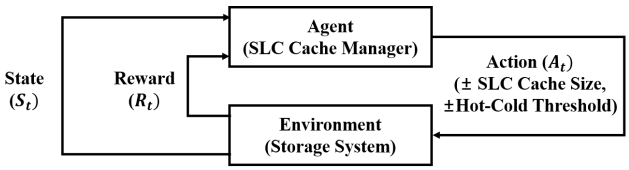


Figure 3: SLC cache management with RL agent.

3 RL-based Dynamic SLC Cache

3.1 Overall Algorithm

The proposed RL-based technique takes an action at every time step. Each time step is defined as a predetermined cumulative amount of host write requests. The size of time step interval determines the sensitivity of the agent's reaction. If the interval is too short, the agent cannot observe meaningful changes on the system states. On the contrary, the agent will miss state changes when the interval is too long. To observe the change on SLC-to-QLC migration cost, each time step is configured to the size of eight SLC blocks in our implementation.

Algorithm 1 shows the pseudo-code of the proposed SLC cache management, which is called at every time step. The current state S_t, the previous state S_{t-1}, and the previous action A_{t-1} are inputs. The algorithm chooses the action A_t for the current state S_t by calling `GetAction()`, which returns the action A which maximizes $Q(S_t,A)$ for the given S_t, by referring to the Q-table. The selected action A_t includes the adjustment on SLC cache size and hot/cold separation threshold. After performing the action A_t, the agent checks the reward of the previous action A_{t-1} in `GetReward()` function. Finally, the agent updates Q(S_{t-1}, A_{t-1}) in the Q-table by using Equation 1.

Algorithm 1 SLC Cache Management

Input: State (S_t), State (S_{t-1}), Action (A_{t-1})
Output: Action (A_t)

1: A_t = GetAction(S_t)
2: Perform A_t
3: R_t = GetReward( )
4: Update Q-value (S_{t-1}, A_{t-1}) with Equation 1

3.2 States

For the agent to learn the optimal policy for the SLC cache management, several states must be observed to know the change of environment, which includes both the host and the SSD subsystem. Table 3 shows the selected states, each of which is divided into multiple bins to limit the total number of different states of the environment.

The SLC cache size is the main state of hybrid SSD. It has a strong influence on garbage collection and SLC-to-QLC migration costs. The SLC cache size is also the target of the

Table 3: States

Category	Information used for State	# of bins
SSD	SLC cache size	9
	Space utilization	4
	Previous action	9
Host Workload	Demand for SLC writes	2
	Update write frequency in SLC cache	2

action. We use nine intervals to represent the SLC cache size. The space utilization is the ratio between the size of valid data and the total capacity of the SSD. As the space utilization becomes higher, the garbage collection cost at the QLC region increases. The agent will learn to restrain itself from increasing the SLC cache when the space utilization is high. The previous action includes the recent decision of the agent and the recent system states. By considering the previous action, the agent can determine a history-aware current action.

For host workload states, we use the demand for SLC writes and the update frequency in the SLC cache. The first one represents how much hot data the host generates, and the second one means the intensity of hotness of the host data written in the SLC cache. If the demand for SLC writes increases, the agent needs to increase the SLC cache or reduce the amount of data to the SLC cache by decreasing the hot/cold separation threshold. The decision must also consider the garbage collection cost in the QLC region. If the update frequency of the data stored in the SLC cache is high, the SLC-to-QLC migration cost will decrease due to many invalidated data in the SLC cache. Therefore, it is also an important factor to determine the SLC cache size suitable for the host workload.

From the number of bins of each state in Table 3, the total number of environment states is 1,296 (= 9×4×9×2×2), and the Q-table size is 5,186 bytes, which can be stored at an SLC flash page with the size of 16 KB. The small Q-table can be cached in the internal DRAM of SSD. If we increase the number of bins of each state, the agent could act more correctly. If the memory space for caching Q-table is insufficient, an on-demand loading technique can be used. However, too many states will require too long training time, and thus the agent cannot be agile. Therefore, considering the trade-off, the number of bins of each state must be determined.

3.3 Reward

The main goal of our algorithm is to improve the overall write performance. The latency of a write request will be different depending on the program mode, i.e., SLC mode or QLC mode. In addition, the host write request to the SLC region can be delayed by SLC-to-QLC migration operations, which can also be delayed by QLC garbage collection operations.

The host write request to the QLC region can also be delayed by QLC garbage collection. Therefore, we need to consider all the costs to calculate the reward of the previous action.

Algorithm 2 shows the pseudo-code of the proposed reward function. It gets several write-related costs and space utilization as inputs. It first calculates the *reclaim cost* and the *host write cost*. The *reclaim cost* is the sum of SLC-to-QLC migration cost ($T_{SLC-to-QLC}$) and QLC garbage collection cost ($T_{QLC-to-QLC}$). The *host write cost* is the sum of SLC cache write cost ($T_{SLCwrite}$) and QLC region write cost ($T_{QLCwrite}$), each of which does not include the waiting time due to SLC-to-QLC migration or QLC garbage collection. The *total write cost* is the weighted sum of the host write cost and the reclaim cost. As the space utilization is higher, a higher weight is given to the reclaim cost and a lower weight to the host write cost. This is because the effect of reclaim cost on the overall performance is more significant at a higher space utilization. By giving a higher weight to the reclaim cost, the reduction on the SLC cache size can be accelerated when the utilization is high. The final reward value is determined to be positive or negative depending on the comparison result to the average total write cost.

Algorithm 2 Reward function

Input: $T_{SLC-to-QLC}$, $T_{QLC-to-QLC}$, $T_{SLCwrite}$, $T_{QLCwrite}$, space utilization U
Output: Reward (R_t)

```
reclaim cost = T_SLC-to-QLC + T_QLC-to-QLC
host write cost = T_SLCwrite + T_QLCwrite
total write cost = (1-U)×host write cost + U×reclaim cost
if total write cost > average total cost then
    R_t = negative reward
else
    R_t = positive reward
end if
Update average total write cost
```

4 Experiments

To evaluate the efficiency of the proposed method, we implemented a trace-driven hybrid SSD simulator. The simulator estimates the latency of each write request by counting the numbers of SLC or QLC flash read, program, and erase operations required to handle the request. Several flash memory parameters were configured, as shown in Table 1. To simplify the analysis, we assumed that the SSD has only one 32GB NAND QLC flash memory chip which consists of 2,138 blocks. However, the proposed algorithm can be applied to more complex SSD architectures by designing a proper write cost model. The page size is 16 KB, and a block has 256 pages and 1024 pages at SLC mode and QLC mode, respectively. The over-provision area for garbage collection is 3% of total capacity. The SSD has 144 KB of write buffer for handling host writes and garbage collection.

Table 4: Workload Characteristic

Trace		PC	Phone	TPC-C	OLTP	LinkBench	YCSB-A
Address space (MB)		1,029	7,606	4,622	5,694	4,482	30,241
Total write amount (MB)		46,426	81,833	39,506	25,866	38,391	97,294
Avg. request size (KB)		66.7	42.8	34.3	35.8	28.2	896.3
Write request size distribution (%)	$\leq$ 128KB	29.47	25.22	53.25	53.4	61.76	0.09
	$\leq$ 256KB	23.08	1.47	2.06	5.01	3.21	0.06
	$\leq$ 512KB	27.03	1.76	16.05	12.42	15.58	3.87
	> 512KB	20.42	71.55	28.64	29.17	19.45	95.98

The hybrid SSD simulator includes a flash translation layer (FTL), which manages 4KB page-level logical-to-physical address mapping. We assumed that the address mapping table is fully cached at DRAM, and thus there are no additional flash operations to access the mapping table. When the number of free blocks of each region is below 5 blocks, SLC-to-QLC migration or garbage collection are invoked.

Since the RL agent runs within the hybrid SSD, it will consume some amount of CPU cycles and DRAM space to maintain Q-table, which may delay the host IO request handling. However, the CPU and DRAM overheads are much smaller compared to the cost of flash memory operations. Therefore, we ignored the overhead of the agent in the simulation.

We used six workloads for simulator inputs collected with the blktrace or the diskmon tool. The PC trace was extracted from a Windows 10-based desktop while running web browsers and compiler. The Phone trace was collected from a smartphone under multiple applications [14]. The TPC-C trace was extracted while executing TPCC-MySQL tool with 160 warehouses on MySQL for 10 minutes. The OLTP trace and Linkbench trace were collected by running Sysbench and Linkbench benchmark on MySQL, respectively. The YCSB-A trace is the workload type A of YCSB running on RocksDB with 16 million keys. Table 4 shows the characteristics of the traces.

We compared the proposed RL-based technique with two previous dynamic SLC techniques: utilization-aware self-tuning (UST) [13] and dynamic write accelerator (DWA) [3,6]. The UST allocates a different number of SLC or QLC physical blocks to each logical block depending on its locality. At maximum, MAX_{SLC} number of SLC blocks can be allocated to a logical block. In order to observe data locality, all write requests are first sent to SLC blocks. When there is no sufficient space in the SLC blocks, the garbage collection for the SLC blocks moves hot data within SLC blocks (SLC-to-SLC), and moves cold data to QLC blocks (SLC-to-QLC). Therefore, as the number of hot logical blocks increases, many SLC blocks will be allocated. So, the total number of SLC blocks will change depending on workloads. The value of MAX_{SLC} is 6 blocks in our experiment. DWA adjusts the SLC cache size according to the space utilization. Considering the characteristics of the target workloads, we used the Setting 1 in Table 2. The hot/cold threshold, θ, is fixed to 32 KB in DWA.

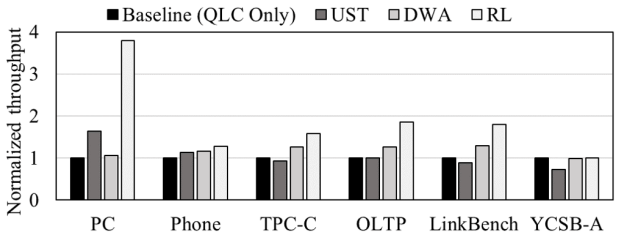
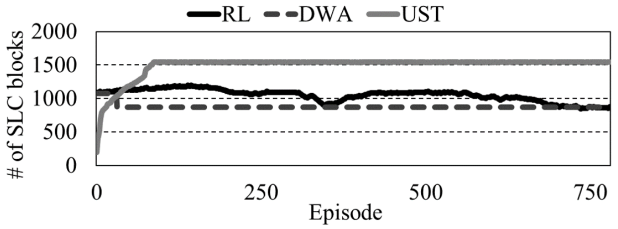
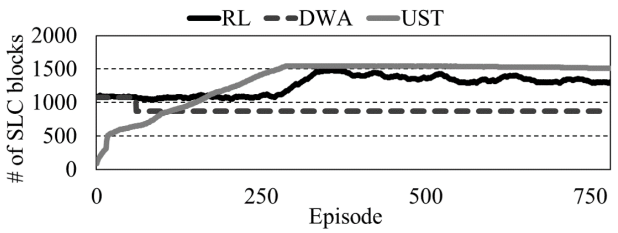


Figure 4: Write throughput comparison.



(a) PC



(b) OLTP

Figure 5: The change of the number of allocated SLC blocks.

4.1 Performance Evaluation

Figure 4 compares write throughputs of different schemes with six workloads. The results are normalized to that of the baseline method, which uses only QLC blocks without the SLC cache. The RL scheme outperforms all other techniques under most workloads except for the case of YCSB-A. Since most of the write requests are large and most of the data are cold in YCSB-A, the SLC cache was little utilized in the RL-based scheme. So, its performance is similar to the baseline performance.

To analyze the behavior of SLC cache management techniques, we observed the number of allocated SLC blocks and

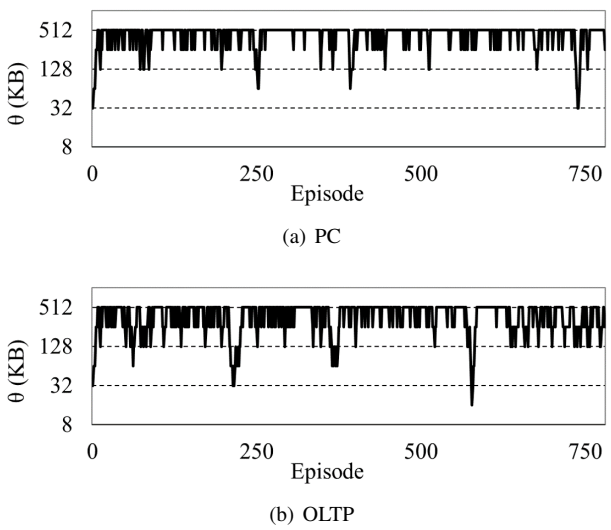


Figure 6: The change of hot/cold separation threshold (θ) at the RL-based SLC cache scheme.

the hot/cold separation threshold (θ) at every 128 MB of host write (episode), as shown in Figure 5 and Figure 6. Compared to DWA and UST, the RL-based technique adjusts more dynamically the SLC cache size and the hot/cold separation threshold. In the PC workload, which has many frequently-updated data, the proposed method allocates less number of SLC blocks than UST does, but maintains a large value of θ, i.e., 512 KB, to store as much hot data as possible in the SLC cache.

Figure 7 shows the breakdown of I/O execution time. The RL technique reduced the QLC-to-QLC or SLC-to-SLC garbage collection overhead compared to UST. Note that UST performs the garbage collection within SLC blocks for hot data whereas our hybrid SSD does not have the SLC-to-SLC migration. The migration and garbage collection costs of the RL-based technique are 65.2% lower than that of UST. Compared to DWA, the QLC write overhead is reduced by the RL scheme. From this result, we can know that the RL-based technique can find the optimal SLC cache parameters under various workloads. Since the RL-based technique reduces migration and garbage collection costs, the write amplification factor (WAF) is also improved. The proposed scheme reduced the average WAF by 20.3% over other techniques.

4.2 Effect of Agent Pre-training

The proposed RL-based technique requires a long training time for unexamined workloads or SSD subsystems. However, if an agent that has been pre-trained at one system is used at other systems, it can adapt to the new environments in a short time. To check the effect of the trained agent, we first trained an agent with several workloads of PC, Phone, YSCB-A, and LinkBench. And then, we observed the performance

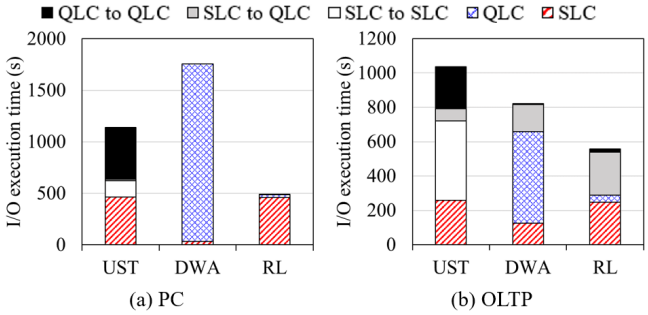


Figure 7: Write execution time comparison.

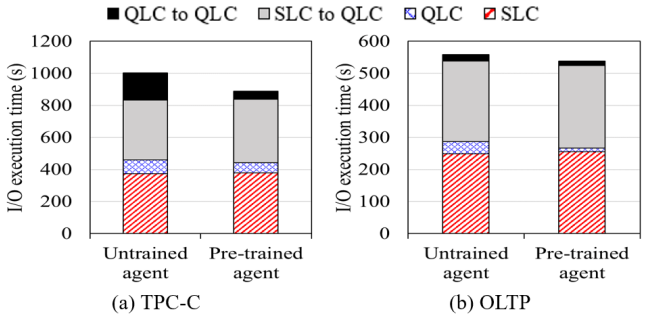


Figure 8: Comparison between pre-trained and untrained agents.

of the agent at other workloads of TPC-C and OLTP. Figure 8 compares I/O performances of pre-trained agent and untrained agent. The pre-trained agent improves the write performance by up to 12.8% over the untrained agent. Therefore, we can know that the RL-based scheme can be applied quickly to a new system by using a pre-trained agent.

5 Conclusion

Due to the increased bit density of flash memory, the hybrid SSD technology is becoming more and more important to enhance write performance. In this paper, we proposed an RL-based SLC cache technique for the hybrid SSDs. The proposed method dynamically determines the optimal SLC cache parameters based on the system states, including the characteristics of the workload and the internal status of the SSD. As a result, the write performance is significantly enhanced without any prior knowledge about host workload or storage characteristics.

As a future work, we have a plan to examine the effect of the proposed RL scheme at a real SSD subsystem. Another work is to apply the technique at multi-stream SSDs [9]. The host can deliver data separation decisions to a multi-stream SSD via the stream interface. Different streams tend to have different update patterns. Thus, we can utilize the stream information to determine the optimal number of SLC blocks of each stream by observing the data lifetime of the stream.

References

[1] Crucial P1 SSD, accessed on Dec. 12, 2019 [Online]. https://www.crucial.com/usa/en/storage-ssd-p1.

[2] Samsung 860 QVO SSD, accessed on Dec. 12, 2019 [Online]. https://www.samsung.com/semiconductor.

[3] The Intel SSD 660p SSD Review : QLC NAND Arrives For Consumer SSDs, accessed on Dec. 12, 2019 [Online]. https://www.anandtech.com/show/13078/the-intel-ssd-660p-ssd-review-qlc-nand-arrives.

[4] Li-Pin Chang. A hybrid approach to NAND-flash-based solid-state disks. *IEEE Transactions on Computers*, 59(10):1337–1349, 2010.

[5] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154, 2010.

[6] Dave Glen. Optimized client computing with dynamic write acceleration. *Micron*, 5, 2014.

[7] Soojun Im and Dongkun Shin. ComboFTL: Improving performance and lifespan of MLC flash memory using SLC flash buffer. *Journal of Systems Architecture*, 56(12):641–653, 2010.

[8] Xavier Jimenez, David Novo, and Paolo Ienne. Software controlled cell bit-density to improve NAND flash lifetime. In *Proceedings of the 49th Annual Design Automation Conference*, pages 229–234. ACM, 2012.

[9] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. The Multi-streamed Solid-State Drive. In *6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'14)*, 2014.

[10] Sungjin Lee, Dongkun Shin, Young-Jin Kim, and Jihong Kim. LAST: locality-aware sector translation for NAND flash memory-based storage systems. *ACM SIGOPS Operating Systems Review*, 42(6):36–42, 2008.

[11] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.

[12] Yoshiki Takai, Mamoru Fukuchi, Reika Kinoshita, Chihiro Matsui, and Ken Takeuchi. Analysis on Heterogeneous SSD Configuration with Quadruple-Level Cell (QLC) NAND Flash Memory. In *2019 IEEE 11th International Memory Workshop (IMW)*, pages 1–4. IEEE, 2019.

[13] Ming-Chang Yang, Yuan-Hao Chang, Chei-Wei Tsao, and Chung-Yu Liu. Utilization-aware self-tuning design for TLC flash storage devices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(10):3132–3144, 2016.

[14] Deng Zhou, Wen Pan, Wei Wang, and Tao Xie. I/O characteristics of smartphone applications and their implications for eMMC design. In *IEEE International Symposium on Workload Characterization*, pages 12–21, 2015.