HotR: Alleviating Read/Write Interference with Hot Read Data Replication for Flash Storage

Suzhen Wu[‡], Weiwei Zhang[‡], Bo Mao[†✉], Hong Jiang[*]
[‡]Computer Science Department of Xiamen University, Xiamen, Fujian, China
[†]Software School of Xiamen University, Xiamen, Fujian, China
[*]Department of Computer Science & Engineering at the University of Texas-Arlington, USA
✉Corresponding author: maobo@xmu.edu.cn

***Abstract*—The read/write interference problem of flash storage remains a critical concern under workloads with a mixture of read and write requests. To significantly improve the read performance in face of read/write interference, we propose a Hot Data Replication scheme for flash storage, called HotR. HotR utilizes the asymmetric read and write performance characteristics of flash-based SSDs and outsources the popular read data to a surrogate space such as a dedicated spare flash chip or an over-provisioned space within an SSD. By servicing some conflicted read requests on the surrogate flash space, HotR can alleviate, if not entirely eliminate, the contention between the read requests and the on-going write requests. The evaluation results show that HotR improves the state-of-the-art scheme in the system performance and cost efficiency significantly. Consequently, the tail-latency of the flash-based storage systems is also reduced.**

***Index Terms*—Flash Storage, Read/Write Interference, Hot Read Data Replication, Performance Evaluation**

I. Introduction

Currently there are different types of NAND flash chips in the storage market [5]. Earlier Single-Level Cell (SLC) NAND flash chips and Multi-Level Cell (MLC) NAND flash chips have been widely commercialized. Recently, Triple-Level Cell (TLC) flash chips have been commercialized, which divide the voltage range into eight windows, representing a 3-bit value. Quadruple-Level Cell (QLC) flash chips, storing a 4-bit value per cell, are recently released by Micron [13]. Encoding more bits per cell increases the capacity of the SSD without increasing the chip size, yet it also decreases reliability by making it more difficult to correctly store and read the bits. As it moves from SLC through MLC to TLC and on to QLC, NAND flash chips are offering higher density and lower cost but at the expense of lower performance and endurance. The asymmetric read and write performance characteristics also induce the read/write interference problem [1], [9], [14], [16].

Figure 1(a) shows the read and write latency for different types of NAND flash chips. The performance gap between read and write accesses are widened significantly along the evolution path of flash chips from SLC to QLC, with increasing read and write interference problem [9], [14]. This interference refers to the phenomenon when on-going write requests preempt available flash memory resource to block read requests, which can cause significant read delays. Such read and write interference results in unpredictable performance and creates significant challenges in consolidated and enterprise environments where read and write requests are mixed. Unfortunately, this read/write interference problem has not received adequate attention. Many existing studies focus on the garbage-collection (GC) induced performance degradation problem because GC operations are much more expensive for flash storage than for other types of storage [6], [7], [20], [18]. However, our evaluations and analysis reveal that the read and write interference is also very harmful for read performance because the write requests are much more frequent than the GC operations.

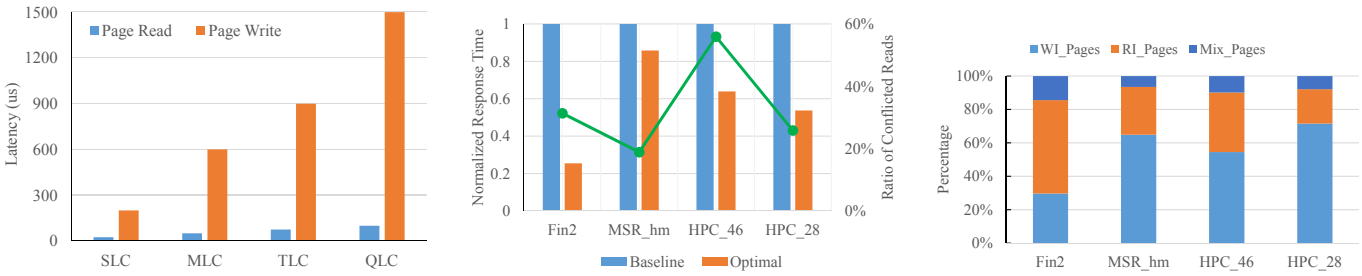
On the other hand, previous studies [10] and our workload analysis suggest that application accesses are performed on either read-only or write-only pages. Here we define three page-access patterns. First, if the vast majority (e.g., 90% or more) of the accesses to a data page are read (write) requests, the page is characterized as *read-only* (*write-only*). Second, if the accesses to a data page are interleaved with reads and writes, the page is characterized as *mixed-page*. Moreover, mixed-pages account for only a small portion of the total pages. Based on the above observations, we propose a Hot Data Replication scheme for flash storage, called HotR, to alleviate the read/write interference problem. HotR utilizes the asymmetry in read and write performance of flash storage and outsources the popular read data pages to a surrogate space such as a dedicated spare flash chip within an SSD. By servicing some conflicted read requests on the surrogate flash space, HotR can alleviate, if not entirely eliminate, the contention between the read requests and the on-going write requests. The evaluation results show that, compared with the state-of-the-art approaches, HotR improves the system performance and cost efficiency significantly. Consequently, the tail-latency of the flash-based storage systems is also reduced.

The rest of this paper is organized as follows. Background and motivation are presented in Section II. We describe the design of hot data replication in Section III. The performance evaluation is presented in Section IV. We conclude this paper in Section V.

II. Background and Motivation

A. Read/Write interference problem

Besides erase operations, write operations are the performance bottleneck of flash storage, not only because they



(a) Latency of different types of flash chips (b) The read/write interference problem (c) Workload characteristics

Fig. 1. The preliminary performance observations and workload characteristics for the motivation of the study.

require more time than read operations, but also because they block the subsequent read operations in the waiting queue [3], [10], [11]. Under a concurrent workload with a mixture of read and write requests, the on-going write requests preempt available flash storage resource so as to block read requests, which we call the read/write interference problem. Such read/write interference results in unpredictable performance and creates significant challenges for flash storage design.

Figure 1(b) shows what percentage of read requests are blocked by the on-going write requests and how the system performance is degraded by the read/write interference, under the four different workloads summarized in Table III. The baseline system is the default configuration with no optimizations and the optimal system refers to one where all the interfered read requests are serviced without being blocked by any of the interfering write requests. An average of 33.0%, with a maximum of 55.9% read requests are blocked by the on-going write requests, resulting of an average of 42.8% performance degradation, and as much as 74.6% for the Fin2 workload. These results indicate that the read/write interference problem significantly degrades system performance and should be carefully addressed when designing flash-based storage systems.

B. Related work

Solutions proposed to address the read/write interference problem fall into two broad categories, I/O scheduler based and redundancy based schemes. Among the most representative in the I/O scheduler based solutions, FIOS [12] schedules requests with the awareness of read/write interference of SSDs. Congming Gao *et al.* proposed PIQ [8] to minimize the access interferences among I/O requests in one batch by exploiting the rich parallelism of SSD. However, all these I/O schedulers only separate read and write requests in batches and the read/write interference problem still exists between read and write request batches. To avoid these drawbacks, Wu *et al.* [17] proposed a P/E (program/erase) suspension scheme, a device-level scheduler, to reduce the write-induced interference. However, frequently suspending/resuming the on-going P/E operation introduces system overhead and suppresses the write performance. Moreover, P/E suspension requires hardware modification.

As an alternative to scheduler-based solutions, redundancy-based solutions add space redundancy to separate interfered reads from writes to minimize the impact of interference. Recently, Skourtis et al. proposed Rails [14] that utilizes the mirrored redundancy to provide predictable read performance under read/write mixed workloads by physically separating reads from writes. However, the space overhead of Rails is quite high due to the replication redundancy. In contrast, TTFlash [20] explores the parity redundancy among flash chips within SSDs to alleviate the GC-induced read performance variability by read reconstruction. The idea of TTFlash may be applicable to alleviating write-induced read performance degradation. However, our experimental results indicate that TTFlash is harmful due to the increased number of internal read requests on flash chips incurred by the read reconstruction, which worsens the read/write interference problem, as revealed in the performance evaluation section IV. Moreover, both Rails [14] and TTFlash [20] address the read/write interference problem by adding and reorganizing the low-level flash device resources, but without exploiting the high-level workload characteristics.

C. Workload characteristics and motivation

Understanding the workload characteristics is important for the design of storage systems. For this purpose, we categorize and differentiate flash pages into the following three distinctive types based on how data stored in a page is accessed by the workloads:

(1) Read-intensive data page (RI): If the vast majority of the accesses (>90%) to a data page are read requests, this page is considered read intensive;
(2) Write-intensive data page (WI): If the vast majority of the accesses (>90%) to a data page are write requests, this page is considered write intensive;
(3) Mixed data (MIX): If the accesses to a data page are neither RI nor WI, i.e., interleaved with reads and writes, this page is considered mixed.

Figure 1(c) shows the distribution of the three types of data pages for the four different workloads listed in Table III in Section IV. Most read requests access the read-intensive data pages and most write requests access the write-intensive data pages. Only an average of 9.6% of the pages are mixed data pages. These observations are also consistent with those reported in the previous studies [10], [19]. More importantly, these workload characteristics imply that the hot read-intensive data pages are not frequently updated by write requests.

With the evolution of flash chips from SLC to MLC, TLC, and QLC, the performance gap between the read and write

TABLE I
COMPARISON BETWEEN HOTR AND THE STATE-OF-THE-ART SCHEMES.

Scheme	Main Idea	Implementation	Space overhead
PIQ [8]	Separates reads and writes in batches and exploits parallelism	Host	No
FIOS [12]			
P/E suspension [17]	Suspends the on-going P/E operations	Hardware modification	
Rails [14]	Separates reads from writes by replication	Host	High
TTFlash [20]	Read reconstruction under parity redundancy	Controller	Moderate
HotR	Hot read data replication on surrogate flash chips	Controller	Moderate

accesses are widened substantially. Thus, the read performance will not only be significantly degraded but also become unpredictable by the on-going write requests. This, combined with awareness of the workload characteristics, motivates the design of HotR that proactively migrates the hot read-intensive data pages to a pre-reserved space (*e.g.*, a staging space such as a dedicated flash chip or an over-provisioned space within an SSD) to significantly alleviate, if not entirely eliminate, the contention between the on-going write requests and the incoming read requests. This helps simultaneously improve the user I/O performance and reduce tail latency of flash-based storage systems. Table I provides a high-level comparison between HotR and the state-of-the-art schemes. Our proposed HotR scheme is orthogonal to and can be easily incorporated into the existing system level I/O schedulers to further alleviate the read/write interference problem for flash-based SSDs.

III. HOTR

A. System overview of HotR

Figure 2 shows a system overview of our proposed HotR within a flash-based SSD. In our current design, HotR is incorporated into the Flash Translation Layer of the SSD. However, HotR can also be implemented in other system software layers, such as the I/O scheduler and RAID controller layers for flash-based storage systems [12], [19].

As shown in Figure 2, HotR consists of five key functional modules: Hot Data Identification, On-Demand Data Migrator, Conflict Detection, Request Distributor, and Administration Interface. *Hot Data Identification* is responsible for monitoring the popularity of read data pages to help *On-Demand Data Migrator* that migrates popular read data pages to and manages the data layout of the redirected data in the staging space. The staging space can be a dedicated flash chip or the over-provisioned space within an SSD and we call both *staging space* in the rest of the paper, as indicated in red in Figure 2. *Conflict Detection* is responsible for detecting the conflicted read requests blocked by the on-going write requests being processed in flash chips. *Request Distributor* is responsible for redirecting the user I/O requests to the proper places, either the original addresses or ones in the staging space, according to the Conflict Detection module. The original addresses in SSDs are referred to as *flash chips* in the rest of the paper, as indicated in blue in Figure 2.

B. Hot data identification and migration

Identifying the hot read data pages is very important for HotR to migrate the popular read data pages in advance to alleviate the read/write interference. To capture both the

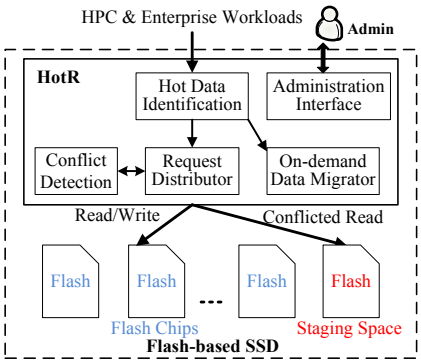


Fig. 2. System overview of HotR.

recency and frequency access patterns of the user applications, we designed and implemented the R-LRU data structure, an LRU-style list in Hot data Identification module, as shown in Figure 3. R-LRU identifies the popular read data pages on each flash chips of SSD and stores the metadata information of the most recently read data pages, including the D-offset, R-offset and Count values defined below.

- *D-offset* indicates the read data page's offset on the flash chips. It is initialized when the read data page is identified as a hot read data page and inserted into the R-LRU list.
- *R-offset* indicates the read data page's offset in the staging space. It is filled as NULL when the entry is initialized and inserted into the R-LRU list. It is updated when the read data page is requested and migrated to the staging space. Thus it also indicates whether the read data page is migrated to the staging space or not.
- *Count* indicates how many times the read data page has been accessed to capture the access frequency pattern.

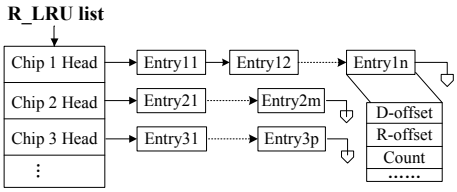


Fig. 3. The R-LRU data structure.

Based on R-LRU, the most recently and frequently accessed read data pages can be identified and proactively migrated to the staging space. Moreover, the popular read data pages are classified by their locations within each flash chip of the SSD. It is easy to query and determine whether a conflicted read request should be serviced in the staging space if an on-going write request is located in the same flash chip.

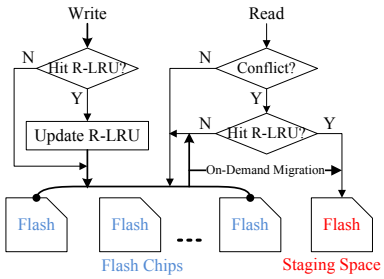


Fig. 4. The workflows of processing write and read requests in HotR.

Since the capacity of the staging space is limited, not all popular read data pages are migrated. In our current design, the percentage of the popular data pages that can be migrated is configured through the *Administration Interface*. Moreover, when the popular read data pages of an SSD are requested by user applications, they are concurrently migrated to the staging space and R-LRU is accordingly updated with *R-offset* filled as the read data page's offset in the staging space by the On-Demand Data Migrator module. Thus, the migration overhead of popular data pages is reduced without affecting the system performance of the flash storage.

C. Request processing workflow

Figure 4 shows the processing workflows for write and read requests. In HotR, all write requests are serviced by the flash chips. In order to ensure data consistency, the R-LRU list is also checked and queried. If the write request hits an entry in the R-LRU list and the R-offset is not NULL, that entry is deleted from the R-LRU list to make sure that the subsequent read requests fetch the up-to-date data pages. The corresponding read data pages in the staging space are marked as invalid simultaneously. If the R-offset is NULL, it indicates that the read data page is not migrated to the staging space. No matter the write requests hit R-LRU or not, the data consistency of the subsequent read requests are not affected because the migrated read data pages in the staging space are always up-to-date and identical to the data pages stored on the flash chips.

Due to the buffer and I/O scheduler optimizations, the write requests arriving at the device level are usually bursty [12]. The large number of write requests and long program latency make most flash chips busy and block the incoming read requests. The Conflict Detection module monitors the program status of the flash chips and returns the results indicating if the incoming read requests are blocked. If a read request is blocked, the Request Distributor module queries the R-LRU list to check whether the read request hits in the R-LRU list. If it hits and the corresponding R-offset is not NULL, the read request is serviced by the staging space and the corresponding count value is incremented by 1. If it hits and the corresponding R-offset is NULL, the read request is serviced by the flash chips. After the read data pages are fetched from the flash chips and returned to the upper layer, HotR writes them to the staging space simultaneously and updates the R-offset and Count values of the entry in the R-LRU list. For the non-hit read requests, entries are initialized and inserted to the R-LRU list for the purpose of identification of hot read data pages.

TABLE II
THE DEFAULT SSD MODEL PARAMETERS.

Parameter	Value	Parameter	Value
Total Capacity	80GB	Planes Per Package	8
Reserved Free Blocks	15%	Blocks Per Plane	2048
Minimum Free Blocks	5%	Pages Per Block	64
Cleaning Policy	Greedy	Page Size	4KB

TABLE III
THE ENTERPRISE AND HPC-LIKE WORKLOAD CHARACTERISTICS.

Traces	Write Ratio	Total I/Os	Average Request Size
Fin2	17.6%	450,068	4.6KB
MSR_hm	72.9%	999,997	8.3KB
HPC_46	60.0%	499,999	6.3KB
HPC_28	80.0%	999,997	6.3KB

IV. PERFORMANCE EVALUATION

A. Evaluation setup and methodology

To evaluate the efficacy of our proposed HotR scheme, we have implemented a prototype of the HotR scheme by integrating it into an open-source SSD simulator developed by Microsoft Research (MSR) [1]. The values of the SSD-specific parameters used in the simulator and performance characteristics of different NAND flash types are summarized in Table II and Figure 1(a) respectively. By default, the TLC-based NAND flash chip is used in the experimental evaluation.

We use both the realistic enterprise-scale workloads and the synthetic HPC-like workloads. The Financial2 workload is collected from OLTP applications running at a large financial institution [15] and the other enterprise-scale workloads were collected from the Microsoft Cambridge Research [4]. The main characteristics of these workloads are summarized in Table III. The synthetic HPC-like workloads allow us to flexibly vary parameters such as read/write ratios, inter-arrival time of requests, read access probability, and sequentiality. We compare the HotR scheme with the original system without any optimizations (Baseline), the Rails [14] and TTFlash [20] schemes in terms of average response time and cost-latency product that provides a measure of cost-effectiveness.

B. Performance results and analysis

Figure 5 shows the average response times, normalized to that of the baseline system with no optimizations, driven by the four workloads. By default, Rails uses 4+4 RAID1 configuration and TTFlash uses 4+1 RAID5 configurations. 10% hot read data pages are migrated in HotR. First, the HotR scheme reduces the average response time of the baseline scheme by up to 54.0% with an average of 21.8%. The significant performance improvement comes from the fact that an average of 17.8% user read requests are redirected to the staging space, as indicated in Figure 6. Most of these migrated user read requests would have been blocked by the on-going write requests without migration, resulting in the contention between read requests and writes requests being significantly alleviated. From the point of view of read requests, the average

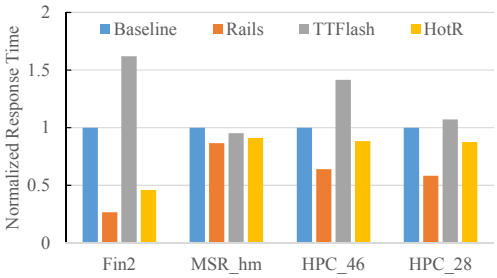


Fig. 5. The normalized average response times driven by the four workloads, normalized to that of the baseline system with no optimizations.

response time is notably reduced by avoiding much of the long latency caused by programming operations (writes).

Second, HotR performs much better than TTFlash but worse than Rails. This is because, while avoiding the interference between user read requests and write requests by read reconstruction in parity-based redundancy, TTFlash's read reconstruction generates a large number of internal read requests that worsen the read/write interference problem. As a result, TTFlash is not extendable to nor suitable for addressing the read/write interference problem for flash storage. By contrast, Rails can not only eliminate the read/write interference problem, but also take advantage of read balance offered by replication redundancy. The percentages of redirected read requests by the Rails and HotR schemes shown Figure 6 suggest that not only all the interfered read requests but also some other read requests are redirected by Rails. However, the significant performance improvement achieved by Rails by leveraging replication redundancy comes with a high space overhead. Previous studies [2] have shown that replication is not cost effective for flash-based storage, which is consistent with our cost-effectiveness results.

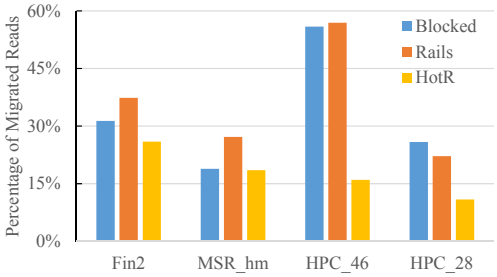


Fig. 6. The percentages of redirected read requests by the Rails and HotR schemes. The Blocked indicates the percentage of read requests that are blocked in the baseline system with no optimizations.

To investigate the impact of the HotR scheme on tail latency under the different workloads, we plot the I/O latencies at the 95^{th}, 97^{th}, 98^{th}, 99^{th}, and 99.5^{th} percentiles in Figure 8. We can see that HotR consistently and significantly outperforms the baseline system in the tail latency performance. While tail latency is mainly caused by the programming operations of flash chips, the write-induced read performance degradations are alleviated by the read request redirections in the HotR scheme. As a result, the percentage of requests with long latency is reduced accordingly. Figure 8 also shows that Rails is the best and TTFlash is the worst in terms of tail latency

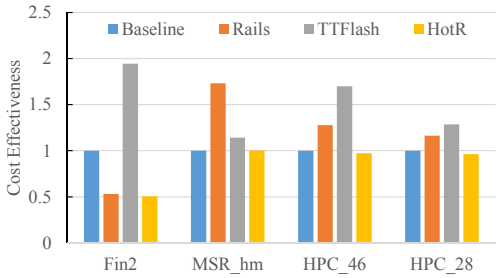
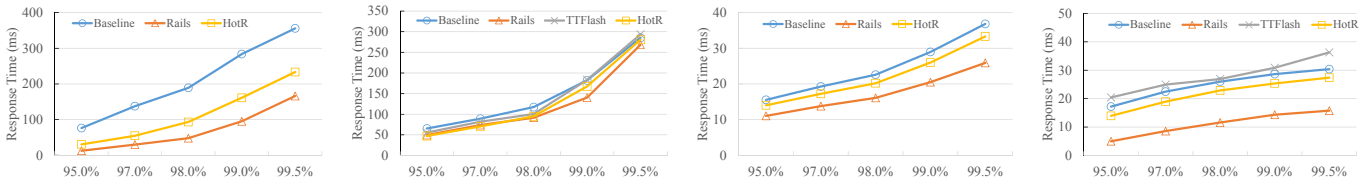


Fig. 7. The cost-effectiveness results of the difference schemes under the four workloads, normalized to the baseline system with no optimizations and space redundancy. *Note that the lower the value, the better for the cost effectiveness.*

among all the schemes. The reason is that Rails can eliminate almost all read/write interferences by data replication. However, TTFlash's read reconstruction generates a large number of internal read requests that worsen the read/write interference problem. For example, the response time of TTFlash is more than 500ms, which is too large to be shown in the Figure 8 under the Fin2 and HPC_46 workloads.

To meaningfully estimate and quantify the cost-effectiveness of HotR in comparison with the state-of-the-art redundancy-based schemes, we use the cost-latency product as a measure for cost-effectiveness, taking inspiration from the energy-latency product that is commonly used in the computer architecture literature to quantify energy efficiency. The cost is mainly on the extra storage space that each scheme consumes, normalized to the baseline system without any extra space overhead. The lower the cost-latency product value of a scheme, the more cost-effective the scheme is. Figure 7 shows cost-effectiveness, in terms of the cost-latency product, of the different schemes based on the latency and cost results obtained from the trace-driven experiments presented earlier in this section.

HotR is clearly the most cost effective, outperforming Rails and TTFlash by 31.6% and 65.7% respectively, and achieving even better cost-effectiveness than that of the baseline system by 13.9%. The reasons behind HotR's superiority in the cost-effectiveness measure are two-fold. First, by exploiting workload characteristics, only hot read data pages are migrated to the staging space that cost much smaller space overhead than the Rails and TTFlash schemes. Although not all the conflicted read requests are migrated, an average of 63.0% conflicted read requests are avoided by HotR. Second, by placing hot read data pages in the staging space, the access latency of the conflicted read requests is reduced significantly. These conflicted read requests are serviced without further interference from write requests, thus reducing their access latency and improving system performance. TTFlash avoids the interference between user read requests and user write requests by read reconstruction that generates a large number of internal read requests. These increased internal read requests can conflict with the user write requests, thus offsetting the advantage of read reconstruction and worsening the read/write interference problem. While TTFlash has smaller space overhead than Rails, its performance is poorer



(a) Fin2 (b) MSR_hm (c) HPC_46 (d) HPC_28

Fig. 8. Tail latencies for different schemes under the four workloads, where the X-axis shows the percentile and the Y-axis indicates the percentile tail latency. *Note that for Fin2 and HPC_46 workloads, the response times of TTFlash are more than 500 ms, too large to be shown in the Figure.*

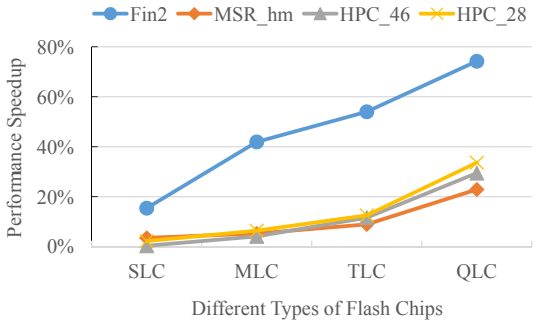


Fig. 9. The performance speedup results of the HotR scheme over the baseline system with 10% read data pages being hot and migrated, as a function of the different flash chips from SLC, to MLC, TLC and QLC, driven by the four workloads.

than Rails. Consequently, neither Rails or TTFlash is cost effective and they are even worse than the baseline system without any optimizations. By contrast, HotR has a better balance between system performance and space overhead by exploiting the workload characteristics, thus achieving the best cost effectiveness among all the schemes.

The performance of HotR scheme can be affected by the types of NAND flash chips. Figure 9 shows the performance speedup results of HotR scheme over the baseline system with 10% read data pages being hot and migrated, as a function of the different flash chips from SLC, to MLC, TLC and QLC, driven by the four workloads. As bit-per-cell density increases from SLC to QLC, the performance speedup achieved by HotR increases accordingly. The reason is that TLC- and QLC-based flash chips have longer program latency, as indicated in Figure 1(a), which significantly prolongs the latency of the conflicted read requests. Moreover, longer program latency also causes larger percentage of read requests being blocked by the on-going write requests. On the other hand, we also see that HotR improves system performance for SLC- and MLC-based flash storage, though the performance is not significant as that for TLC- and QLC-based flash storage.

V. Conclusion

This paper proposes HotR to address the read/write interference problem of flash storage by exploiting both the workload characteristics and device characteristics. HotR scheme removes some conflicted read requests from the original addresses in flash chips by proactively replicating the frequent read data pages into the staging space. The extensive trace-driven experimental results show that HotR significantly improves the system performance and cost efficiency of the state-of-the-art approaches. Consequently, the tail-latency of the flash-based storage systems is also reduced.

Acknowledgement

This work is supported by the National Natural Science Foundation of China under Grant No. 61872305, No. 61772439, No. U1705261, and No. 61472336, the US National Science Foundation under Grant No. CCF-1704504 and No. CCF-1629625.

References

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. Davis, M. Manasse, and R. Panigrahy. Design Tradeoffs for SSD Performance. In *USENIX ATC'08*, Jun. 2008.
[2] M. Balakrishnan, A. Kadav, V. Prabhakaran, and Dahlia Malkhi. Differential RAID: Rethinking RAID for SSD Reliability. In *EuroSys'10*, Apr. 2010.
[3] G. Belelloch, J. Fineman, P. Gibbons, Y. Gu, and J. Shun. Sorting with Asymmetric Read and Write Costs. In *SPAA'15*, Jun. 2015.
[4] Block I/O Traces in SNIA. http://iotta.snia.org/tracetypes/3.
[5] Y. Cai, S. Ghose, E. Haratsch, Y. Luo, and O. Mutlu. Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives. *Proceedings of the IEEE*, 105(9):1666–1704, 2017.
[6] W. Choi, M. Jung, and M. Kandemir. Parallelizing Garbage Collection with I/O to Improve Flash Resource Utilization. In *HPDC'18*, Jun. 2018.
[7] J. Cui, Y. Zhang, J. Huang, W. Wu, and J. Yang. ShadowGC: Cooperative Garbage Collection with Multi-level Buffer for Performance Improvement in NAND Flash-based SSDs. In *DATE'18*, Mar. 2018.
[8] C. Gao, L. Shi, M. Zhao, C. Xue, K. Wu, and E. Sha. Exploiting parallelism in I/O scheduling for access conflict minimization in flash-based solid state drives. In *MSST'14*, May 2014.
[9] S. Koh, C. Lee, M. Kwon, and M. Jung. Exploring System Challenges of Ultra-Low Latency Solid State Drives. In *HotStorage'18*, Jun. 2018.
[10] Q. Li, L. Shi, C. Xue, K. Wu, C. Ji, Q. Zhuge, and E. Sha. Access Characteristic Guided Read and Write Cost Regulation for Performance Improvement on Flash Memory. In *FAST'16*, Feb. 2016.
[11] B. Mao and S. Wu. Exploiting Request Characteristics and Internal Parallelism to Improve SSD Performance. In *ICCD'15*, Oct. 2015.
[12] S. Park and K. Shen. FIOS: A Fair, Efficient Flash I/O Scheduler. In *FAST'12*, Feb. 2012.
[13] QLC. https://www.micron.com/products/advanced-solutions/qlc-nand.
[14] D. Skourtis, D. Achlioptas, N. Watkins, C. Maltzahn, and S. Brandt. Flash on Rails: Consistent Flash Performance through Redundancy. In *USENIX'14*, Aug. 2014.
[15] UMass Trace Repository. http://traces.cs.umass.edu/index.php/.
[16] H. Wang, J. Zhang, S. Shridhar, G. Park, M. Jung, and N. Kim. DUANG: Fast and Lightweight Page Migration in Asymmetric Memory Systems. In *HPCA'16*, Mar. 2016.
[17] G. Wu and B. He. Reducing SSD Read Latency via NAND Flash Program and Erase Suspension. In *FAST'12*, Feb. 2012.
[18] S. Wu, Y. Lin, B. Mao, and H. Jiang. GCaR: Garbage Collection aware Cache Management with Improved Performance for Flash-based SSDs. In *ICS'16*, Jun. 2016.
[19] S. Wu, W. Zhu, G. Liu, H. Jiang, and B. Mao. GC-aware Request Steering with Improved Performance and Reliability for SSD-based RAIDs. In *IPDPS'18*, May 2018.
[20] S. Yan, H. Li, M. Hao, H. Tong, S. Sundararaman, A. Chien, and H. Gunawi. Tiny-Tail Flash: Near-Perfect Elimination of Garbage Collection Tail Latencies in NAND SSDs. In *FAST'17*, Feb. 2017.