VIOS: A Variation-Aware I/O Scheduler for Flash-Based Storage Systems

Jinhua Cui[1], Weiguo Wu[1](✉), Shiqiang Nie[1], Jianhang Huang[1], Zhuang Hu[1], Nianjun Zou[1], and Yinfeng Wang[2]

[1] School of Electronic and Information Engineering, Xi'an Jiaotong University, Shaanxi 710049, China
cjhnicole@gmail.com, wgwu@xjtu.edu.cn

[2] Department of Software Engineering, ShenZhen Institute of Information Technology, Guangdong 518172, China

Abstract. NAND flash memory has gained widespread acceptance in storage systems because of its superior write/read performance, shock-resistance and low-power consumption. I/O scheduling for Solid State Drives (SSDs) has received much attention in recent years for its ability to take advantage of the rich parallelism within SSDs. However, most state-of-the-art I/O scheduling algorithms are oblivious to the increasingly significant inter-block variation introduced by the advanced technology scaling. This paper proposes a variation-aware I/O scheduler by exploiting the speed variation among blocks to minimize the access conflict latency of I/O requests. The proposed VIOS schedules I/O requests into a hierarchical-batch structured queue to preferentially exploit channel-level parallelism, followed by chip-level parallelism. Moreover, conflict write requests are allocated to faster blocks to reduce access conflict of waiting requests. Experimental results shows that VIOS reduces write latency significantly compared to state-of-the-art I/O schedulers while attaining high read efficiency.

Keywords: Process variation · Solid state drive · I/O scheduling · Flash memory · Parallelism

1 Introduction

As NAND flash storage capacity becomes cheaper, NAND flash-based SSDs are being regarded as powerful alternatives to traditional Hard Disk Drives (HDDs) in a wide variety of storage systems [1]. However, SSDs introduce an out-of-place update mechanism and exhibit asymmetric I/O properties. In addition, a typical SSD usually offers rich parallelism by consisting of a number of channels with each channel connecting to multiple NAND flash chips [2,3]. Most of conventional I/O schedulers including NOOP, CFQ and Anticipatory are designed to mitigate the high seek and rotational costs in mechanical disks, leading to many barriers to take full advantage of SSDs. Thus, I/O scheduling for SSDs has received much

G.R. Gao et al. (Eds.): NPC 2016, LNCS 9966, pp. 3–16, 2016.
DOI: 10.1007/978-3-319-47099-3_1

attention for its ability to take advantage of the unique properties within SSDs to maximize read and write performance.

Most of existing I/O scheduling algorithms for SSDs, such as PAQ [4], PIQ [5] and AOS [6], focus on avoiding resource contention resultant from shared SSD resources, while others take special consideration of Flash-Translation-Layer (FTL) [7] and garbage collection [8]. These works have demonstrated the importance of I/O scheduling for SSDs to reduce the number of read and write requests enrolled in conflict, which are the major contributors to access latency. However, little attention has been paid to dynamically optimize the data transfer latency, which could naturally reduce the access conflict latency when conflicts are unavoidable anymore.

The capacity of NAND flash memory is increasing continuously, as a result of technology scaling from 65 nm to the latest 10 nm technology and the bit density improvement from 1 bit per cell to the latest 6 bits per cell [9,10,20]. Unfortunately, for newer technology nodes, the memory block P/E cycling endurance has significantly dropped and process variation has become relatively much more significant. Recently many works have been proposed to exploit the process variation from different perspectives. Pan et al. [11] presented a dynamic BER-based greedy wear-leveling algorithm that uses BER statistics as the measurement of memory block wear-out pace by taking into account of inter-block P/E cycling endurance variation. Woo et al. [12] introduced a new measure that predicts the remaining lifetime of a flash block more accurately than the erase count based on the findings that all the flash blocks could survive much longer than the guaranteed numbers and the number of P/E cycles varies significantly among blocks. Shi et al. [13] further exploited the process variation by detecting supported write speeds during the lifetime of each flash block and allocating blocks in a way that hotter data are matched with faster blocks, but they did not reorder the requests in the I/O scheduling layer. Therefore, none of these works focuses on incorporating the awareness of inter-block variation into I/O scheduling to minimize the access conflict latency of I/O requests.

This paper proposes a variation-aware I/O scheduler to exploit the speed variation among blocks for write performance improvement without degrading read performance. The key insight behind the design of VIOS is that a variation-aware scheduler can organize the blocks of each chip in a red-black tree according to their detected write speeds and maintain a global chip-state vector, where the current number of requests for each chip is recorded, so as to identify conflict requests. By scheduling arrived requests into hierarchical-batch structured queues to give channel-level parallelism a higher priority than chip-level parallelism and allocating conflict write requests to faster blocks to exploit inter-block speed variation, access conflict latency of waiting requests is reduced significantly. Trace-based simulations are carried out to demonstrate the effectiveness of the proposed variation-aware I/O scheduling algorithm.

The rest of the paper presents the background and related works in Sect. 2. Section 3 describes the design techniques and implementation issues of our VIOS for flash storage devices. In Sect. 4, experimental evaluation and comparison with several alternative I/O schedulers are illustrated. Section 5 concludes this paper.

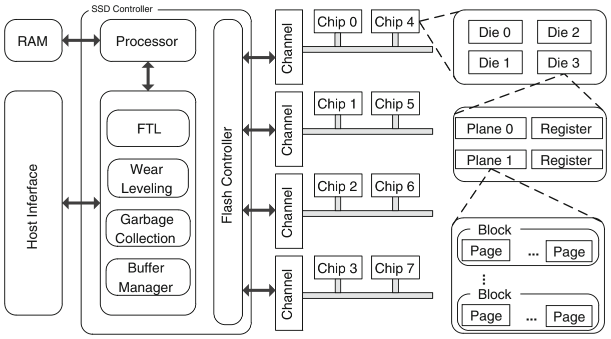


Fig. 1. SSD hardware diagram

2 Background and Related Work

2.1 SSD Organization

Flash memory chips are organized in channels and ways, as shown in Fig. 1. Within each flash memory chip are one or more dies, each further consisting of multiple planes, which are the smallest unit to be accessed independently and concurrently. Each plane is composed of a number of erase units, called blocks, and a block is usually comprised of multiple pages, which are the smallest unit to read/write. There are four main levels of parallelism which can be exploited to accelerate the read/write bandwidth of SSDs. Actually, the importance of exploiting parallelism on read/write performance improvement has been testified by numerous research works from different perspectives. For example, Roh et al. [14] explored to enhance B+-tree's insert and point-search performance by integrating a new I/O request concept (psync I/O) into the B+-tree which can exploit the internal parallelism of SSDs in a single process, Hu et al. [2] argued that the utilization of parallelism, primarily determined by different advanced commands, allocation schemes, and the priority order of exploiting the four levels of parallelism, will directly and significantly impact the performance and endurance of SSDs.

Since the advance command support required by the die and plane level parallelism is not widely supported by most of SSDs, the degree of parallelism is usually governed by the number of channels multiplied by the number of flash memory chips in a channel, without taking the die and plane level parallelism into consideration. In this paper, our VIOS also exploit both channel-level parallelism and chip-level parallelism by scheduling arrived requests into hierarchical-batch structured queues.

2.2 Process Variation of Flash Memory

Along with the bit density developments and technology scaling of NAND flash memory, the aggravating process variation (PV) among blocks has been magnified, which results in largely different P/E cycling endurance within different memory blocks when given the same ECC. PV is caused by the naturally occurring variation in the attributes of transistors, such as gate length, width and oxide thickness, when integrated circuits are fabricated. The distribution of the Bit Error Rates (BER) of flash blocks is characterized as the log Gaussian distribution by measuring 1000 blocks of a MLC NAND Flash memory chip manufactured in 35-nm technology at the same 15K P/E cycles [11].

Besides, there is a close relationship between BER and the speed of write operations. Typically, when program operations are carried out to write data into flash memory cells, the incremental step pulse programming (ISPP) scheme is introduced to appropriately optimize program voltage with the certain step size $\varDelta V_p$ that triggers a trade-off between write speed and BER. Using larger $\varDelta V_p$, fewer steps are used to the desired level, thus the write latency is shorter. As the promising effect of reducing the write latency of write requests, however, the margin for tolerating retention errors is also reduced, resulting in higher BER. Therefore, with the awareness of both process variation and the BER-speed relationship, write speed for lower-BER blocks can be increased at the cost of reduced noise margins, while that for higher-BER blocks should be carefully optimized without exceeding the capability of the deployed ECC. The challenge to detect the proper write speed for each block at its current worn out state is also solved by periodically reading out the written data to find out the number of faulty bits, and the analysis indicates that overhead is negligible [13]. In this paper, the blocks of each chip are sorted according to their detected write speeds and conflict write requests are allocated to faster blocks to reduce access conflict of waiting requests.

2.3 I/O Scheduler for Flash-Based SSDs

An increasing number of I/O scheduling algorithms have been proposed to improve flash memory storage system performance from different perspectives. The first two algorithms for Flash-Based SSDs called IRBW-FIFO and IRBW-FIFO-RP were designed by Kim et al. [15]. The basic idea is to arrange write requests into bundles of an appropriate size while read requests are independently scheduled by taking the read/write interference into consideration. Numerous research works enhanced the IRBW-FIFO and IRBW-FIFO-RP by exploiting rich parallelism in I/O scheduling, such as PAQ [4] and PIQ [5]. In addition, there is also recognition on the importance of fairness in multi-programmed computer systems and multi-tenant cloud systems, such as FIOS [16] and FlashFQ [17].

However, little attention has been paid to optimize the data transfer latency dynamically, which could naturally reduce the access conflict latency when conflicts can not be avoided anymore. Our VIOS focuses on incorporating the awareness of inter-block variation into I/O scheduling to minimize the access conflict latency of I/O requests. Fortunately, all these existing algorithms are somewhat orthogonal to our work, and can be used concurrently with the proposed VIOS to optimize the efficiency of flash-based I/O scheduling.

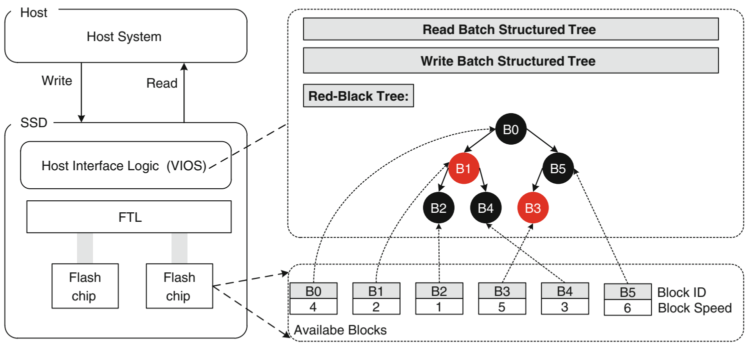


Fig. 2. Organizational view of the VIOS scheduler

3 Details of VIOS

Motivated by the increasingly significant inter-block variation introduced by the advanced technology scaling, a variation-aware I/O scheduler is proposed to minimize the access conflict latency of I/O requests. The proposed VIOS scheduler is implemented in the host interface logic (HIL), where the knowledge of both the interface protocols and SSD specific characteristics contributes to better device-specific scheduling decisions for I/O requests. The main idea behind the design of VIOS is to allocate conflict write requests to faster blocks so as to reduce access conflict latency of waiting requests. To achieve this, our scheduler organizes the blocks of each chip in a red-black tree according to their detected write speeds and maintains a global chip-state vector, where the current number of requests for each chip is recorded, so as to identify conflict requests.

3.1 Block Management

After detecting the proper write speed for each block at its current worn out state by periodically reading out the written data to find out the number of faulty bits, it is important to manage blocks with different write speeds for VIOS to easily schedule requests to appropriate blocks. In VIOS, the red-black tree structure is adopted as the main data structure to sort its blocks in detected speed order. Since the advance command support required by the die and plane level parallelism is not widely supported by most of SSDs and the relatively higher chip-level conflicts are the major contributors to access latency, the blocks of each chip are associated with a red-black tree respectively. Figure 2 shows the main data structures associated with VIOS. Once all the pages of a block have been programmed, it will be set as an invalid node in the red-black tree. When the prepared empty blocks are used up, a time-consuming task called Garbage Collection (GC) is triggered to reclaim stale pages for free write space and then the states of erased blocks become valid again. The blocks are evicted and inserted into another place of the red-black tree only when the write speed detection process is triggered and its ΔV_p is decreased, corresponding to reduced write speed.

3.2 Global Chip-State Vector

To build a scheduling algorithm that can identify conflicts and exploit the speed variation based on the degree of conflicts, we propose a global chip-state vector to track the current number of requests for each chip. The global chip-state vector mainly depends on the location of data, which is determined by different data allocation schemes and the logical sector number (LSN) of I/O requests. Let's take an SSD where the number of chips is *ChipNum* and the size of each page is *PageSize* as an example. For a given static allocation scheme where the priority order of parallelism is channel-level parallelism first, followed by the chip-level parallelism [18], the assemblage of chips accessed by request r can be defined as:

$$A_r = \left\{ \mu | \mu = \left(\frac{(lsn(r)+\lambda)*SectorSize}{PageSize} \right) \% ChipNum \right\} \\ where \quad 0 \le \lambda < len(r) \tag{1}$$

where *lsn(r)* and *len(r)* are the accessed logical sector number and data size in sectors of request r respectively, while *SectorSize* is the sector size in bytes. For a global chip-state vector defined as $(NR_0, NR_1, \ldots, NR_i)$ where NR_i is the current number of requests for chip i, when pushing arrived requests into the queue or issuing chosen requests to SSDs, the *NR* of each chip accessed by requests is updated as follows:

$$NR_i = \begin{cases} NR_i + 1 & arriving \\ NR_i - 1 & issued \end{cases}, i \in A_r \tag{2}$$

3.3 Conflict Optimized Scheduling Mechanism

Next, we propose the conflict optimized scheduling technique, which aims to reduce access conflict latency by exploiting the rich parallelism within SSDs and the speed variation among blocks. It consists of two components: (1) a hierarchical-batch structured queue to avoid conflicts from chip to channel. (2) A variation-aware block allocation technique that assigns conflict write requests to faster blocks to reduce access conflict of waiting requests.

Hierarchical-Batch Structured Queue. Since there are four main levels of parallelism which can be exploited to accelerate the read/write bandwidth of SSDs, the conflicts can also be classified into four types based on the physical resources contended by arrived requests. Among them, channel conflicts and chip conflicts are taken into account for the reason that the advance command support required by the die and plane level parallelism is not widely supported by most of SSDs. In the hierarchical-batch structured queue, the requests are grouped into batches from bottom up based on the chip-level and channel-level conflict detection respectively. Requests in the same chip batch can be serviced in chip-level parallel, while requests belong to the same channel batch can be executed in completely independent channel-level parallel. Each chip batch and channel batch use A_{chip} and $A_{channel}$ respectively to track the assemblage of chips and

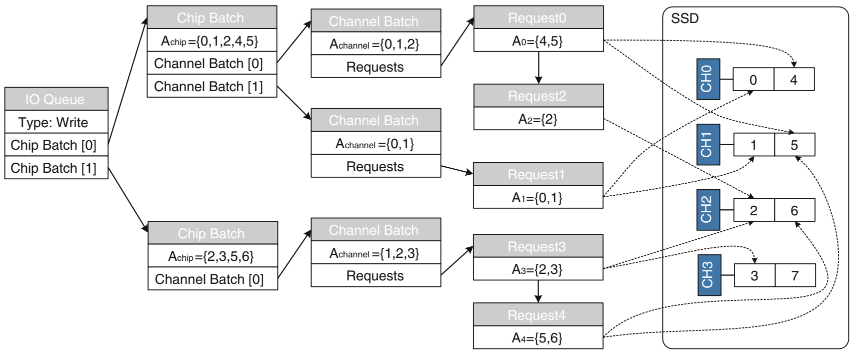


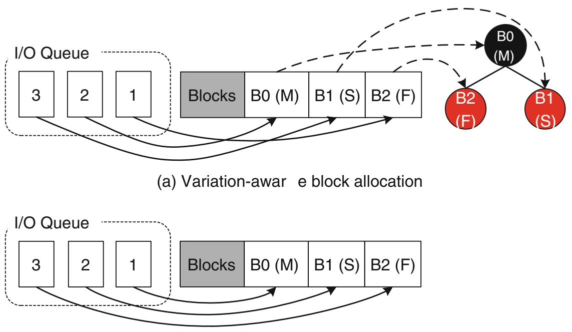
Fig. 3. An example that shows VIOS conflict reduction with the hierarchical-batch structured queue

channels accessed by all its requests. Each time when a new request r arrives, the detection of chip-level conflicts is enabled by repeated intersection operations of A_{chip} with A_r, whose resulting assemblage of conflict chips ($A_{conflict}$) can be modeled as:

$$A_{conflict} = A_{chip} \cap A_r \tag{3}$$

Once a chip batch that has no conflict with r is found, which means $A_{conflict}$ is empty, the request is added to the chip batch and A_{chip} is updated. Otherwise, a new chip batch is created for the new request. After that, the detection of channel-level conflicts is performed in the same manner to further exploit channel-level parallelism of requests within the same chip batch. For example, after scheduling the first five requests into the hierarchical-batch structured queue shown in Fig. 3, if one more request accessing chips $\{6, 7\}$ arrives, it is found that there is no chip-level conflict between the new request and the first chip batch, where the new request is thus added and A_{chip} is updated to $\{0, 1, 2, 4, 5, 6, 7\}$. Then each channel batch of the first chip batch is checked and the second channel batch where the new request has no channel-level conflict is chosen with $A_{channel}$ being updated to $\{0, 1, 2, 3\}$.

Variation-Aware Block Allocation. Motivated by the findings that speed variation among blocks can be easily detected, we propose a variation-aware block allocation algorithm to optimize the data transfer latency dynamically, which could naturally reduce the access conflict latency when conflicts cannot be avoided anymore. Each time when a request is issued, the NR of each chip accessed by the request is checked and then updated. Since a request processing may access multiple chips in NAND flash memory, we scatter the request into separate sub-requests. Each sub-request is only able to gain access to one chip. The sub-request accessing the chip whose NR is more than 1 will be allocated to a faster block from the red-black tree of the chip. Otherwise, a slower block is chosen for the sub-request. Figure 4 shows the process of scheduling three



(b) Normal block allocation

Fig. 4. Block allocation process. Program speed represented by Slow (S), Medium (M) and Fast (F).

conflict sub-requests when only three pages in three blocks with different speeds are empty. As Fig. 4(a) shows, the variation-aware block allocation algorithm assigns the first two sub-requests to currently faster blocks for the reason that one or more sub-requests are still waiting in the queue. The last sub-request is allocated to a slower block because no conflict sub-request is waiting at this moment. Assuming that the write latency of fast, medium and slow blocks are 150 µs, 180 µs, and 210 µs respectively, the average request response time of the proposed algorithm is $(150 + 330 + 540)/3 = 340$ µs, while that of the normal algorithm (Fig. 4(b)) which distributes blocks in order is $(180 + 390 + 540)/3 = 370$ µs. Therefore, by incorporating the awareness of inter-block variation into I/O scheduling, the access conflict latency of I/O requests is reduced significantly.

Overhead Analysis. The overheads of the proposed I/O scheduler are analyzed as follows. According to the detailed descriptions of components above, the implementation of VIOS needs to maintain hierarchical-batch structured queues in the I/O queue. Since the number of channels and chips in NAND flash memory is limited, all sets can be stored as binary words and the set-intersection/set-union operation can be performed as an O(1)-time bitwise-AND/OR operation. This storage overhead is negligible for an I/O queue. Furthermore, the complexity of adding an incoming I/O request into the hierarchical-batch structured queues is proportional to the sum of the number of chip batches and the number of channel batches, and it is less than the queue length of I/O scheduler, which also has negligible cost.

4 Experimental Results

To evaluate our proposed variation-aware I/O scheduler, we perform a series of trace driven simulations and analysis. We implement VIOS as well as baseline NOOP scheduling and state-of-the-art PIQ scheduling within an event-driven simulator named as SSDSim [18], which provides the detailed and accurate simulation of each level of parallelism. Note that write speed detection technique is implemented with all of the schedulers. We simulate a 128 GB SSD with 8 channels, each of which is connected to 8 chips. For the flash micro-architecture configuration, each flash chip employs 2 dies, where each die contains 4 planes and each plane consists of 2048 blocks. Each flash block contains 64 pages with a page size of 2 KB. All these settings are consistent with previous works [5]. Page mapping FTL is configured to maintain a full map of logical pages to physical ones and greedy garbage collection scheme is implemented.

The BER growth rate that follows Bounded Gaussian distribution is used to simulate the process variation of flash memory, where the mean μ and the standard deviation σ are set as 3.7×10^{-4} and 9×10^{-5} respectively [11]. The maximal possible write step size is set to 0.6 and the step of decreasing $\varDelta V_p$ is set to 0.03. We use 600 µs as the 2 bit/cell NAND flash memory program latency when $\varDelta V_p$ is 0.3, 20 µs as memory sensing latency and 1.5 ms as erase time. Four different wear-out stages corresponding to 15K, 12K, 9K and 6K P/E cycles are evaluated. We evaluate our design using real world workloads from MSR Cambridge traces [19] and the write-dominated Financial1 trace [20], where 500000 I/Os of each trace are used in accordance with previous work.

4.1 Performance Analysis of VIOS

Our experiments evaluate scheduling performance with read and write latency. Figure 5 shows the average read latency for NOOP, PIQ and VIOS tested under the P/E cycling of 12K. As can be observed, VIOS improves the average read latency by about 17.66 % compared to NOOP, indicating that the hierarchical-batch structured read queue helps VIOS exploit multilevel parallelism inside SSDs by resolving resource conflicts. However, the improvements in average read latency brought by VIOS are not significantly higher than those obtained when using PIQ. This is because the variation-aware block allocation technique of VIOS mainly serves write request, and read requests are always preferentially scheduled in both PIQ and VIOS without being affected by write performance improvement.

Figure 6 plots the simulation results on average write latency when different scheduling algorithms are used in the variation-induced SSDs. To facilitate the comparison, the average write latency is normalized against the case of using NOOP algorithm. The first thing to observe is that VIOS outperforms NOOP and PIQ with write latency reduction by 22.93 % and by 7.71 % on average, respectively. This is because both the hierarchical-batch structured write queue and the variation-aware block allocation algorithm reduce access conflict of write requests. However, the write performance improvements under different traces vary greatly. For example, compared to PIQ, the greatest improvement made in the *src* trace is 17.17 %, but the slightest improvement made in the *mds* trace is only 2.73 %. This is due to the different percentages of requests enrolled in conflict – VIOS works for I/O intensive applications where more requests can be processed in parallel and optimized. Table 1 shows the percentages of conflicts

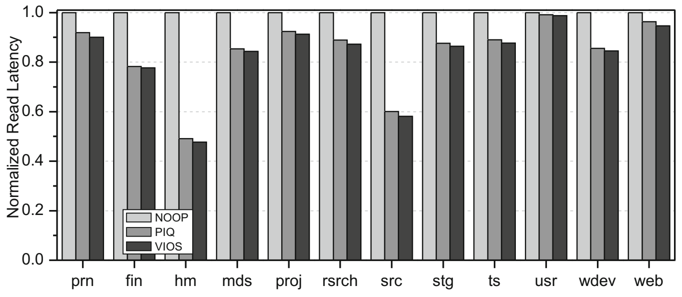


Fig. 5. Average read latencies for three different types of schedulers (normalized to the NOOP scheduler)

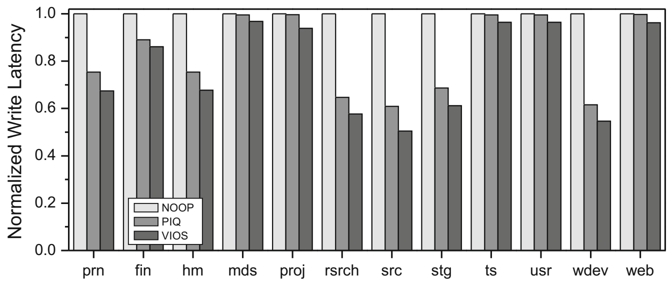


Fig. 6. Average write latencies for three different types of schedulers (normalized to the NOOP scheduler)

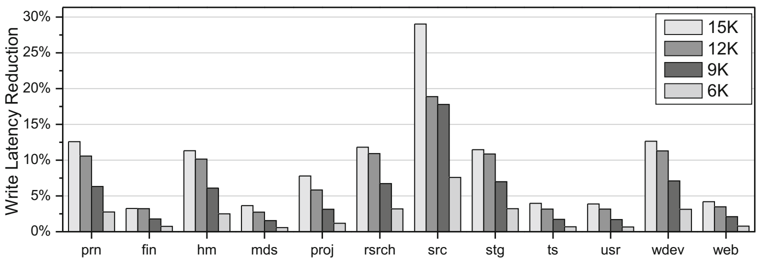


Fig. 7. A comparison of the write latency reduction relative to PIQ among four different wear-out stages

collected under the P/E cycling of 12K with NOOP scheduler. One can also observe that the percentage of conflicts in *src* is 77.74 %, which has an impact on improving efficiency of VIOS. In contrast, that of *mds* is 4.70 %, indicating that fewer conflicts lead to slighter write performance improvement.

Figure 7 gives a comparison of the write latency reduction relative to PIQ among four different wear-out stages. One can observe from these results that the write latency reduction is positive all the time, which means VIOS always outperforms PIQ under four different wear-out stages. Furthermore, VIOS improves

Table 1. Characteristics of used workloads

Traces	Read I/O	Write I/O	ReadConflicts	WriteConflicts	ConflictsRatio
prn	61422	438578	37678	261787	59.89
fin	129342	370658	12135	129379	28.30
hm	139527	360473	54387	168864	44.65
mds	26301	473699	10947	12570	4.70
proj	151549	348451	38615	32852	14.29
rsrch	43803	456197	19413	217828	47.45
src	81377	418623	28146	360549	77.74
stg	43818	456182	21657	210355	46.40
ts	76687	423313	27566	7161	6.95
usr	206983	293017	91391	9068	20.09
wdev	102663	397337	48986	170202	43.84
web	239458	260542	89753	19184	21.79

write performance by 2.25 %, 5.25 %, 7.71 % and 9.63 % on average under the P/E cycling of 6K, 9K, 12K and 15K respectively. As can be observed, with the increased number of P/E cycling, the write performance improvement brought by VIOS gets greater. This is a very reasonable result since BER spread grows as flash memory cells gradually wear out with the P/E cycling, corresponding to more significant variation among blocks, which improves the efficiency of the variation-aware block allocation strategy in VIOS. Overall, these results clearly demonstrate the effectiveness of VIOS in reducing the write latency during the entire flash memory lifetime.

4.2 Sensitivity Analysis of VIOS

To measure the sensitivity of VIOS to the I/O intensity, we repeated our experiments by varying the number of chips and the baseline program latency. Either fewer chips or slower program speeds increase the probability of access conflict. Figure 8 plots the normalized average write latency for each trace under 64, 56, 48, 40 and 32 chips when using VIOS. From the plots, it can be seen that the write latency increases as the number of chips decreases. For most traces, varying the number of chips from 64 to 32 increases the write latency by less than 25 %. However, for traces *src* and *wdev*, the increase in write latency is 59.48 % and 36.09 % respectively. By comparing the results with the percentages of write conflicts shown in Table 1, it can be observed that the increment in average write latency is greater when the number of write conflicts is larger. For example, the write latency of *src* that has most write conflicts (360549) is increased with maximum rate (59.48 %), while these of *mds*, *ts* and *usr* which have fewer write conflicts (12570, 7161 and 9068) are increased with minimum rate (6.05 %, 5.29 % and 4.91 %). On one hand, the number of conflicts is

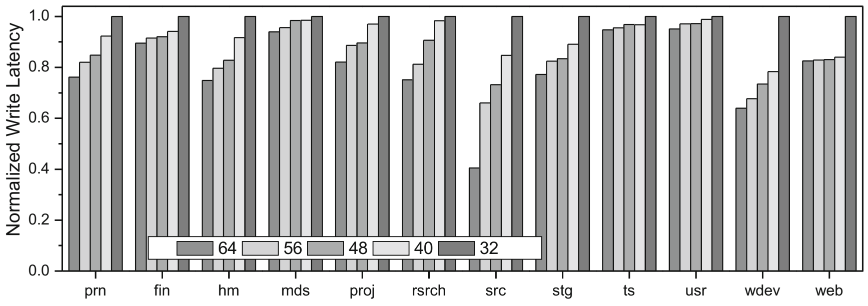


Fig. 8. Average write latencies for five different numbers of chips when using VIOS (normalized to 32 chips)

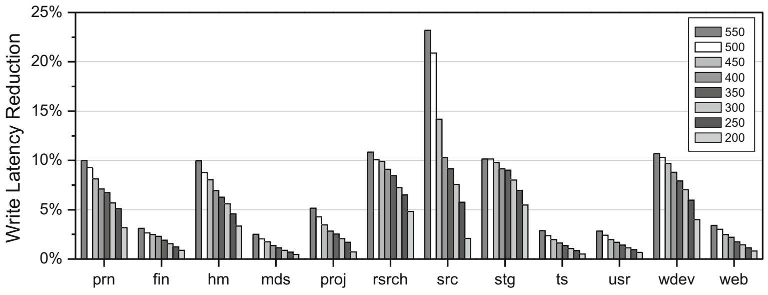


Fig. 9. A comparison of the write latency reduction relative to PIQ among eight different baseline program latencies

proportional to the quotient of access density and the number of chips, which means that more conflicts occur when reducing chips for traces with more intensive I/O. On the other hand, the average write latency is proportional to the square of the number of write conflicts, amplifying the effect of each new conflict.

Figure 9 plots the impact of the baseline program latency on the write latency reduction relative to PIQ. The x-axis is the baseline program latency for $\Delta V_p = 0.3$, varying from 200 µs to 550 µs. The number of conflict requests increases as the program latency is delayed, thus improving the benefit from hierarchical-batch structured queues and variation-aware block allocation technique. However, the effect of program latency delay is greater than that of reduction in the number of chips. For example, from 200 µs to 550 µs, the write latency reduction for *rsrch* varies from 4.83 % to 10.85 %, compared to a slighter variation from 10.92 % to 13.63 % as the number of chips varies from 64 to 32. The major reason is that delaying program latency not only increases the number of conflict requests, but also amplifies the access conflict latency, which is the dominant factor for slow write operations.

5 Conclusion

In this paper, we propose a variation-aware I/O scheduler (VIOS) for NAND flash-based storage systems. The process variation is exploited to reduce the access conflict latency of SSDs when conflicts are unavoidable anymore. VIOS organizes the blocks of each chip in a red-black tree according to their detected write speeds and allocate conflict write requests to faster blocks to exploit inter-block speed variation. In addition, the hierarchical-batch structured queue that focuses on the exploration of the parallelism of SSDs is presented. Furthermore, with diverse system configurations such as wear-out stages, the number of chips and the baseline program latency, VIOS reduces write latency significantly compared to the state-of-the-art NOOP and PIQ while attaining high read efficiency.

Acknowledgment. The authors would like to thank the anonymous reviewers for their detailed and thoughtful feedback which improved the quality of this paper significantly. This work was supported in part by the National Natural Science Foundation of China under grant No. 91330117, the National High-tech R&D Program of China (863 Program) under grant No. 2014AA01A302, the Shaanxi Social Development of Science and Technology Research Project under grant No. 2016SF-428, the Shenzhen Scientific Plan under grant No. JCYJ20130401095947230 and No. JSGG20140519141854753.

References

1. Min, C., Kim, K., Cho, H., Lee, S.W., Eom, Y.I.: SFS: random write considered harmful in solid state drives. In: USENIX Conference on File and Storage Technologies, p. 12, February 2012
2. Hu, Y., Jiang, H., Feng, D., Tian, L., Luo, H., Ren, C.: Exploring and exploiting the multilevel parallelism inside SSDs for improved performance and endurance. IEEE Trans. Comput. **62**(6), 1141–1155 (2013)
3. Chen, F., Lee, R., Zhang, X.: Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing. In: IEEE 17th International Symposium on High Performance Computer Architecture, pp. 266–277, February 2011
4. Jung, M., Wilson III, E.H., Kandemir, M.: Physically addressed queueing (PAQ): improving parallelism in solid state disks. ACM SIGARCH Comput. Architect. News **40**(3), 404–415 (2012)
5. Gao, C., Shi, L., Zhao, M., Xue, C.J., Wu, K., Sha, E.H.: Exploiting parallelism in i/o scheduling for access conflict minimization in flash-based solid state drives. In: 30th Symposium on Mass Storage Systems and Technologies, pp. 1–11, June 2014
6. Li, P., Wu, F., Zhou, Y., Xie, C., Yu, J.: AOS: adaptive out-of-order scheduling for write-caused interference reduction in solid state disks. In: Proceedings of the International MultiConference of Engineers and Computer Scientists, vol. 1 (2015)
7. Wang, M., Hu, Y.: An I/O scheduler based on fine-grained access patterns to improve SSD performance and lifespan. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing, pp. 1511–1516, March 2014
8. Jung, M., Choi, W., Srikantaiah, S., Yoo, J., Kandemir, M.T.: HIOS: a host interface I/O scheduler for solid state disks. ACM SIGARCH Comput. Architect. News **42**, 289–300 (2014)

9. Ho, K.C., Fang, P.C., Li, H.P., Wang, C.Y.M., Chang, H.C.: A 45 nm 6b/cell charge-trapping flash memory using LDPC-based ECC and drift-immune soft-sensing engine. In: IEEE International Solid-State Circuits Conference Digest of Technical Papers, pp. 222–223, February 2013
10. Zuloaga, S., Liu, R., Chen, P.Y., Yu, S.: Scaling 2-layer RRAM cross-point array towards 10 nm node: a device-circuit co-design. In: IEEE International Symposium on Circuits and Systems, pp. 193–196, May 2015
11. Pan, Y., Dong, G., Zhang, T.: Error rate-based wear-leveling for NAND flash memory at highly scaled technology nodes. IEEE Trans. Very Large Scale Integration (VLSI) Syst. **21**(7), 1350–1354 (2013)
12. Woo, Y.J., Kim, J.S.: Diversifying wear index for MLC NAND flash memory to extend the lifetime of SSDs. In: Proceedings of the Eleventh ACM International Conference on Embedded Software, p. 6, September 2013
13. Shi, L.H., Di, Y., Zhao, M., Xue, C.J., Wu, K., Sha, E.H.M.: Exploiting process variation for write performance improvement on NAND flash memory storage systems. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **99**, 1–4 (2015)
14. Roh, H., Park, S., Kim, S., Shin, M., Lee, S.W.: B+-tree index optimization by exploiting internal parallelism of flash-based solid state drives. In: Proceedings of the VLDB Endowment, pp. 286–297 (2011)
15. Kim, J., Oh, Y., Kim, E., Choi, J., Lee, D., Noh, S.H.: Disk schedulers for solid state drivers. In: Proceedings of the Seventh ACM International Conference on Embedded Software, pp. 295–304, October 2009
16. Park, S., Shen, K.: FIOS: a fair, efficient flash I/O scheduler. In: USENIX Conference on File and Storage Technologies, p. 13, February 2012
17. Shen, K., Park, S.: FlashFQ: a fair queueing I/O scheduler for flash-based SSDs. In: USENIX Annual Technical Conference, pp. 67–78, June 2013
18. Hu, Y., Jiang, H., Feng, D., Tian, L., Luo, H., Zhang, S.: Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity. In: Proceedings of the International Conference on Supercomputing, pp. 96–107, May 2011
19. Narayanan, D., Thereska, E., Donnelly, A., Elnikety, S., Rowstron, A.: Migrating server storage to SSDs: analysis of tradeoffs. In: The European Conference on Computer Systems, pp. 145–158, April 2009
20. UMass Trace Repository. http://traces.cs.umass.edu
21. Cui, J., Wu, W., Zhang, X., Huang, J., Wang, Y.: Exploiting process variation for read and write performance improvement of flash memory. In: 32th International Conference on Massive Storage Systems and Technology, May 2016