2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems

DLS: A Delay-Life-Aware I/O Scheduler to Improve the Load Balancing of SSD-based RAID-5 Arrays

Guohua Yan, Renhai Chen*, Qiming Guan, Zhiyong Feng
College of Intelligence and Computing, Shenzhen Research Institute of Tianjin University, Tianjin University
Email: {yanguohua, renhai.chen, GuanQiMing, zyfeng}@tju.edu.cn

***Abstract*—With the advent of the era of big data, efficient reading and writing of data have become the focus of attention. Solid State Drives (SSDs) have been a competitive alternative to traditional hard disks in disk arrays due of their better I/O performance. However, the uniform I/O scheduling strategies used in traditional disk arrays (e.g., FCFS) do not fully consider the characteristics of SSDs read and write performance asymmetry, resulting in insufficient utilization of disk array performance. Furthermore, a typical feature of a new type of solid-state drive based on new flash memory is the limited lifetime. How to balance the load while maximizing the lifetime of the flash SSDs have become the key to disk arrays research.**

To address the above issues, in this paper, we propose a novel delay-life-aware load scheduling strategy called "DLS", which can optimize the I/O performance and prolong the lifetime of disk array. Firstly, due to different access times for different read/write request operations, we intelligently predict the delay time of each read and write request. Then, to improve the life of the disk array, we predict the current life of each SSD based on the load status of each SSD. Finally, based on the load delay and lifetime on each SSD, we design a write distribution model to dynamically predict the allocation address of the next write request. To demonstrate the effectiveness of delay-life-aware load scheduling strategy, we conduct experiments via a trace-driven simulator. The experimental results show that, compared with EGS, the IOPS and request execution time of the DLS scheduling strategy is improved by more than 8% compared with EGS [1]. The proposed scheme can enhance the lifetime of the RAID-5 as its can allocate write requests more balanced than other scheduling strategies.

***Index Terms*—delay-life-aware, request scheduling, solid state disk, disk arrays**

I. Introduction

With the booming of big data applications, high performance, high reliability, low consumption storage devices are widely concerned in data center environments for large data access [2]. Redundant array of independent disks (RAID) is a combination of multiple hard disks into a hard disk array group so that the performance can reach or exceed an expensive hard disk with a large capacity [3]. It is widely concerned because it provides higher performance and data redundancy than a single hard disk. RAID has different levels due to its implementation (e.g., RAID-1, RDIA-2). RAID-5 is widely used in the industry due to its storage performance, data security and storage cost storage solutions [4]. However, due to solid-state drives (SSDs) can provide higher input/output operations per second, lower power consumption, and higher shock resistance [5], [6] than traditional hard-disk drives (HDDs), SSDs have been replacing traditional HDDs in disk arrays [7]. But, the uniform I/O scheduling strategies used in traditional disk arrays (e.g., FCFS) do not fully consider the characteristics of SSDs read and write performance asymmetry [8]–[10], resulting in insufficient utilization of the performance of SSDs, and small random writes will degrade the I/O performance of SSDs [11]. Besides, due to the limited lifetime of the new flash memory SSDs [12], how to maximize the lifetime of SSDs to improve the overall lifetime of the disk array has become the focus of research [13].

Several prior arts have investigated to exploit the several I/O scheduling approaches to disk arrays. Takahashi H et al. propose the I/O balancing scheduler [14] to evenly distribute the I/O access frequency of each disk, which is usually an out-of-order scheduler to improve the balancing of disk arrays. In addition, the pseudo-parallel I/O scheduling [15] is proposed to a closed queuing request model based on a RAID system with cache. Parallelism is applied to key aspects of RAID system internal migration. The above scheduling algorithm has the following two problems for an SSD-based disk array. On the one hand, the uniform I/O scheduling strategies used in traditional disk arrays(e.g., FCFS) do not fully consider the characteristics of SSDs read/write performance asymmetry, resulting in insufficient utilization of disk array performance. On the other hand, The SSD has not been fully considered to have a limited service lifetime, resulting in a decrease in the overall lifetime of the entire disk array. However, the recently proposed the Effective Global I/O Scheduler [1] policy only gives different weights to read/write requests and does not fully consider read/write requests execution status in SSD to read/write performance asymmetry. Due to the request delay for each read and writes request in each type is different, the idea of using only the type as the weight of the request delay is not comprehensive enough.

To address this issue, we propose a novel delay-life-aware load scheduling strategy, which can optimize the I/O performance and prolong the lifetime of disk array. On the one hand, we fully consider the organization of the parallel components in SSDs and investigate the utilization of these components to linearly predict the delay of each request. On the other hand, to improve the overall lifetime of the disk array, we dynamically predict the lifetime of each SSD and use this as one of the metrics to guide the request allocation. Finally, based on the two criteria for each SSD's request delay and its remaining

*Corresponding author.

978-1-7281-2058-4/19/$31.00 ©2019 IEEE
DOI 10.1109/HPCC/SmartCity/DSS.2019.00341

lifetime, we use a write distribution model to dynamically allocate the storage space for the next write request. Note that since the address of each read request is fixed, and we only use it to update the delay load of its SSD.

The main contributions are summarized as follows:

- For each read and writes request, we linearly predict every request execution delay and then predict the total latency of each SSD. Besides, we dynamically predict the lifetime of each SSD based on the load of the SSD.
- We propose a novel delay-life-aware load scheduling strategy, which can dynamically allocate a suitable SSD for the next write request based on the latency and lifetime factors of each SSD using a write distribution algorithm.
- We use a trace-driven simulator to evaluate the effectiveness of the proposed scheme. The novel delay-life-aware load scheduling strategy can improve load balancing and prolong the lifetime of disk arrays.

The rest of this paper is organized as follows. In Section II, we discuss the background of the internal architecture of SSDs, existing scheduling strategy in disk arrays. In Section III, we present the design of the novel delay-life-aware load scheduling strategy. Then, the experimental results are presented in Section IV. Finally, Section V concludes this work. Finally, for the sake of easy presentation and better comprehension, we summarize the definition of main notations used in the whole paper in Table I.

II. Background and Motivation

A. SSD internals

Fig. 1 shows the internals of a NAND-flash package, which is organized as a hierarchical and parallelized structure [16]. A host interface connects to the host through an interface connection (e.g., SATA or IDE bus). The buffer is mainly used to cache read and write data. In most SSDs, multiple channels (e.g., 2-10) are connected to the flash controller, and each channel connected more than flash. Inside the flash memory contains multiple components (e.g., die, plane). By

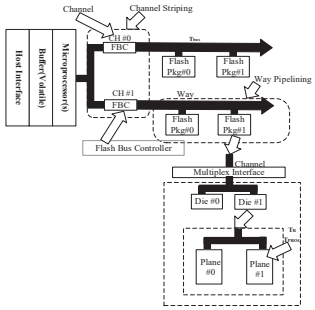


Fig. 1. The internal structure of the SSD

understanding the internal architecture of SSD, we can divide the parallel architecture of SSD into system-level parallelism and flash-level parallelism [17]. and operations at each level can be parallelized or interleaved to predict the request delay. System-level parallel scheduling includes parallelism at the channel-level and the flash-level. It can be seen that the channel bus is one of the bottlenecks of system-level parallel scheduling. Flash is a stand-alone unit that executes as a request, and internally contains two parallel components, ie die, plane. Full use of Die parallel and Plane parallel can effectively improve the performance of flash and improve the throughput of SSD.

We consider the organization of the parallel components in SSDs and investigates the utilization of these components. Then, we studied the SSD's characteristics and will affect its service time are abstracted into a set of parameters. The parameters include the number of channels, the number of flash packages in each channel, the data access time in the bus, etc. A linear model is built to dynamically predict the delay of the request based on the execution state of the underlying SSD and the parameters we abstracted.

B. The structure and example of the SSD-based RAID-5

Solid State Drives (SSDs) offer higher input/output operations per second, lower power consumption, and higher impact resistance than traditional hard disk drives (HDDs), so they are proper to be utilized in disk arrays. We take an SSD RAID-5 as an example, as shown in Fig. 2. The entire RAID-5 is divided into strips (e.g., Strip#0), each strip containing multiple chunks (e.g., D0, P0). The chunks in the strip are distributed among all SSDs in the RAID-5 to provide system-level fault tolerance. Each stripe has a parity block (e.g., P0) that is calculated based on the XOR operation of the data blocks in the same strip. Parity blocks in RAID-5 are typically stored in different SSDs in a round-robin (RR) fashion for load balancing. When updating data chunks, their corresponding parity chunks are also updated by read-modify-write (RMW) or read-rebuild-write (RRW). This parity chunks update in RAID-5 will increase additional I/O and reduce the performance and durability of SSD in RAID. Finally, we let chunks size equal to the physical page size of SSD in this paper.

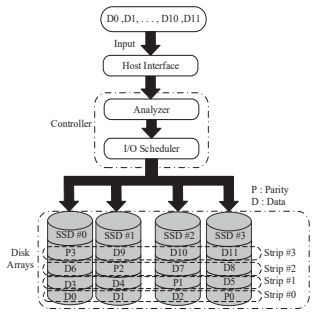


Fig. 2. System structure and data layout of SSD-based RAID-5

C. The existing scheduling strategy

In order to improve the efficiency of SSD-based RAID-5, many scheduling methods have recently been proposed. Next we will explain these scheduling strategies.

TABLE I
DEFINITION OF NOTATIONS USED IN SECTION III.

Notation	Definition
T_{bus}	The time of transmitting one byte on the data bus
S_{page}	The physical page size in SSDs
$T_{commands}$	The time of transmitting commands on the data bus
$N_{request\_batch}$	The number of handling read/write requests at a time
T_{data}	The time of transmitting a physical page on the data bus
$T_{channel\_w}$	The time at which write data is transmitted on the channel
$T_{channel\_r\_commands}$	The time at which the read request commands are transmitted on the channel
$T_{channel\_r\_data}$	The time of transmitting multiple physical pages from flash to the SSD controller over the channel
T_R	The time of reading the physical page to the cache register
T_{PROG}	The page program (write) time from register
T_{flash_w}	The time of handling write requests in the flash
T_{flash_r}	The time of handling read requests in the flash
$T_{request\_arrive}$	The time at which the request arrived the SSD controller
$T_{request\_end}$	The request completion time
$T_{request\_time}$	The request handling time
$delay_i$	The delay of the ith batch requests
$channel\_delay_n$	The total delay of the nth channel
SSD_delay	The delay time of the SSD
SSD_use_life	The limited lifetime of the SSD
SSD_life	The remaining life of the SSD
$SSD\_capacity$	The capacity of the SSD
$W\_capacity$	The average data size written in the SSD per day
$W\_totalCapacity$	The total size of data written in the SSD
P/E_count	The flash rewritable times of the SSD
avg_l	The average life of disk array
avg_d	The average delay of disk array
L_i	The remaining life of the i-th SSD
D_i	The delay of the i-th SSD

- First Come First Served (FCFS): Using a request queue to cache read and write requests, FCFS always dispatches the request that is currently at the top of the ready queue to the running state. It is a non-preemptive strategy, but poor performance may result in high latency and a reduction in the overall life of the disk array.
- Effective Global I/O Scheduler (EGS): EGS optimizes I/O performance and prolongs the life of SSD-based RAID-5 arrays. Depending on the access time of the read/write operations, It is separate the read and writes operations to give different weights, and each disk has the same number of windows (consisting of several read/write I/Os). On the other hand, in order to avoid unbalanced write allocations, we have devised an additional strategy to extend the life cycle of the SSD.
- Round-Robin Scheduling (RR): since the request addresses for all read requests are fixed. The round-robin (RR) dispatching uniformly distributes to write requests to all channels in the SSD. The RR request scheduling policy can be considered fair from the fact that all write requests are evenly distributed to each SSD.

III. DESIGN OF DELAY-LIFE-AWARE LOAD BALANCING SCHEDULING STRATEGY

In this section, we will introduce a delay-life-aware Load Scheduling strategy. First, we will introduce a model based on the delay-life-aware Load balancing Scheduling Strategy, after which we give the details of the algorithm.

A. Scheduling model

1) Design of delayed linear prediction model: In SSD-based multiple parallel systems, in order to fully utilize the parallel resources, it is critical to construct a model so as to effectively and efficiently predict the resource utilization of SSD devices. In the process, the SSD's characteristics that affect its service time are abstracted into a set of parameters. The parameters include the number of channels, the number of flash packages in each channel, the data access time in the bus, etc. Next, we present the proposed model in detail.

The parallel resources inside SSDs include channel-level parallelism and flash-level parallelism. We first present the time consumption used in each SSD channel according to Eqs. (1) and (4). Then, the data access time used in multiple dies/planes can be represented by using Eqs. (6) and (7). The transmission time of the batch write requests in the flash channel is calculated as follows:

$$T_{channel\_w} = N_{request\_batch} * (T_{commands} + T_{data}) \quad (1)$$

$T_{commands}$ and T_{data} can be calculated via Eqs. (2) and (3) respectively as follows:

$$T_{commands} = 7 * T_{bus} \quad (2)$$

$$T_{data} = S_{page} * T_{bus} \quad (3)$$

The calculation of $T_{commands}$ in Eq. (4) is divided into three parts. First, a T_{bus} time is used to transmit a start command (00h) over the bus. After that, it takes 5 T_{bus} times to transfer the address of the read/write operation over the bus, followed by a T_{bus} time transfer end command (30h). The T_{data} is the time used to transfer a physical page on the bus. The transmission time of the batch read requests in the flash channel is calculated as follows:

$$T_{channel\_r\_commands} = N_{request\_batch} * T_{commands} \quad (4)$$

$$T_{channel\_r\_data} = N_{request\_batch} * T_{data} \quad (5)$$

Eq. (4) represents the time used to transfer the commands and Eq. (5) represents the time used to transfer the batch data requests.

In the flash chip, the execution time of read and write requests can be represented by the following equations:

$$T_{flash\_w} = N_{request\_batch} * (T_{commands} + T_{data}) + T_{PROG} \quad (6)$$

$$T_{flash\_r} = N_{request\_batch} * (T_{commands} + T_{data}) + T_R \quad (7)$$

T_R and T_{PROG} are the read and write time in the plane, respectively. For write requests, the die interleaving command sends multiple requests to the die. When the write requests are issued to the die, multiple write requests are sent to the cache register. Then, the flash memory begins to program data. The time at which the flash executes the write requests is shown in Eq. (6). However, for reading requests, the die interleaving command sends multiple read requests from different plane addresses to each die. When the read requests are issued to the die, multiple read request addresses are sent to the plane. Then, the read data is cached in the cache register. The time consumed to handle the read requests is shown in Eq. (7).

Eq. (8) is used to calculate the total delay of the nth channel, which is described as follows:

$$channel\_delay_n = \sum_{i=1}^{N} delay_i \tag{8}$$

The Eq. (9) for calculating $delay_i$ is presented as follows:

$$delay_i = T_{request\_end} - T_{request\_arrive} - T_{request\_time} \tag{9}$$

The $T_{request\_arrive}$ indicates the time when the request arrived, $T_{request\_time}$ indicates the time when the request was executed, $T_{request\_end}$ indicates the time at which the execution of the request ends. The delay time of the SSD is as shown in Eq. (10).

$$SSD\_delay = \max_{1<x<C} channel\_delay_i \tag{10}$$

2) design of life-aware model: A typical feature of a new flash-based SSD is its limited lifetime, which is related to the P/E_count of flash. The SSD life prediction is shown in Eq. (11).

$$SSD\_use\_life = \frac{SSD\_capacity * P/E\_count}{W\_capacity * 365} \tag{11}$$

From Eq. (11), we can use Eq. (12) as the budget for the life of the SSD.

$$SSD\_life = \frac{(SSD\_capacity(GB) * P/E\_count) - W\_totalCapacity}{W\_capacity} \tag{12}$$

$W\_capacity$ and $W\_totalCapacity$ need to consider the write amplification of flash.

3) design of write distribution scheduling model: The average delay and the average life are calculated as shown in Eqs. (14) and (13), and the lowest delay time of the SSD is as shown in Eq. (15)

$$avg\_l = \frac{\sum_{i=1}^{S} SSD\_life_i}{S} \tag{13}$$

$$avg\_d = \frac{\sum_{i=1}^{S} SSD\_delay_i}{S} \tag{14}$$

$$SSD\_min\_delay_i = \min_{1<x<C} channel\_delay_i \tag{15}$$

As shown in Fig. 3, we describe in detail the request allocation process for the write distribution model. We first calculate the average delay avg_l and the average lifetime avg_d of the entire disk array, and then select all SSDs with delay less than avg_l and lifetime greater than avg_d, finally, find an SSD with the lowest delay.

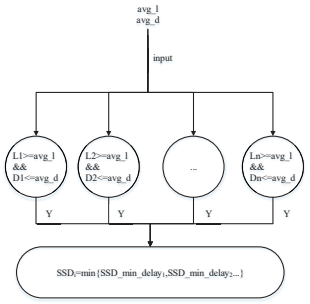


Fig. 3. Request allocation process of write distribution

B. Delay-Life-Aware load balancing scheduling algorithm

Algorithm 1: Delay-Life-Aware Load balancing Scheduling Algorithm

```
   Input :
         Requests from the client
   Output:
         Requests after the execution path is assigned
1  if requests(0).type == 1 then
2        /*Obtain the average delay avg_l and the average lifetime avg_d by Eqs. (13) and (14)*/
3        s ← select_SSD_by distribution model (avg_l,avg_d);
4  /*Asynchronously update the lifetime and delay of the current SSD*/
5  if requests(0).type == 1 then
6        /*Obtain the channel with the least delay
7        according to Eq. (8)*/
8        c ← select_channel (channels);
9        /*Get the free flash of the current channel*/
10       f ← select_flash (c.flashs);
11       /*In order to make full use of the parallel structure inside the SSD,
12       we create the write request group*/
13       requestGroup ← build_request_group (requests);
14 else
15       s, c, f ← obtain_eexecutePath(r);
16       /*In order to make full use of the parallel structure inside the SSD,
17       we create the read request group*/
18       requestGroup ← build_request_group (requests, requestsCount);
19 end
20 /*update the delay and lifetime of the current SSD by Eqs. (10) and (12)*/
21 update_SSD_state (s, c, f, requestGroup);
```

Algorithm 1 describes the proposed delay-life-aware Load balancing Scheduling Algorithm. When the requests are accumulated in the SSD disk array, algorithm 1 will all the received requests are assigned to the SSD queue according to some special rules. Then, the request in the SSD queue is waiting to be executed. Since the access location of the read requests is fixed, we only need to allocate the flash space for the write requests. The procedures of request scheduling are presented as follows. First,the delay-based scheduling determines the type of request. In the algorithm, we use 1 to represent the write type (line 1). We select an SSD with a delay less than avg_l and a lifetime greater than avg_d to process the next request(line 3). After that, the current SSD life and delay are predicted asynchronously (line 21).

IV. Evaluation

In this section, we first describe the experimental setup for evaluating the effectiveness of the proposed delay-based scheduling strategy. Then, we comprehensively evaluate and analyze the experimental results of the proposed scheme.

A. Experimental setup

We develop a trace-driven RAID-5 simulator to evaluate different scheduling strategies. In the RAID-5 simulator, we

carry out the experiments respectively on 4 disks and 8 disks RAIDS. We set 16 channels and 8 flash chips per channel (128 flash chips in total) in every SSD device. The transfer speed of buses inside SSDs is configured to be 50 MB/s. The page read latency of MLC NAND is configured to be 20 us or 30us, and the page write latency is configured to be 200 us or 400 us. The page size that we used in this experiment is 2 KB. The block consists of 128 pages and each plane contains 2,048 blocks. The SSD flash uses dual-die and two-plane architecture. We evaluate the effectiveness of the proposed scheme on the Financial and WebSearch data sets [18]. We examine the impact of different transfer speed of bus and different page write latency on different strategies.

TABLE II
CHARACTERISTICS OF THE BENCHMARKS

	DATA SIZE (MB)	Write Fraction (%)	Avg. Write Size (KB)	Avg. Read Size (KB)
Financial1	16944	82.3	1.6	1.4
Financial2	9284	22.4	1.4	1.3
WebSearch1	62108	0.01	1.7	1.8
WebSearch3	62996	0.02	1.6	1.8

Table II presents the characteristics of the benchmarks used in our experiment, including request size and read/write ratio. We choose real enterprise workloads, which contain the read-intensive web search (e.g., WebSearch1) and write-intensive financial transactions (e.g., Financial1).

We evaluate three strategies in our experiment:

- Round-Robin Scheduling (RR): A classical scheduling method in computer systems, which dispatch uniformly distributes write requests to all channels in the SSD.
- Effective Global I/O Scheduler (EGS): This scheduler tries to balance workload and writes on different disks in SSD-based RAID-5.
- Delay-Life-Aware Load balancing Scheduling (DLS): This scheduler tries to balance minimum latency and maximum lifetime in SSD-based RAID-5.

B. Experimental results

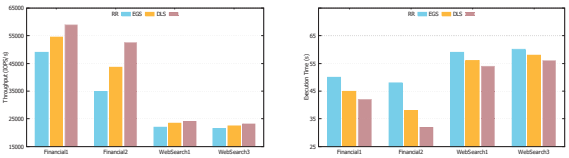
In this section, we give numerical results when using different schedules. We use RAID-5 with 4 disks and 8 disks respectively to evaluate the performance of RR, EGS, and DLS, we use IOPS (Input/Output Operations Per Second), execution time and write frequencies as our metrics. In addition, we explore the impact of the page read and write latency.

1) Performance comparison of three strategies:

First, we configured the page read and write latency to be 20.0 us and 200 us. Fig. 4(a) and Fig. 4(b) show the result compared with RR and EGS scheduling strategy when using RAID5 with 4 SSD, we can find that for write-intensive traces, the IOPS and request execution time of the DLS scheduling strategy are improved more than 20.2% and 8.0% respectively compared with the RR and EGS scheduling strategy. However, for read-intensive traces, the IOPS and request execution time of EGS and DLS are roughly the same.

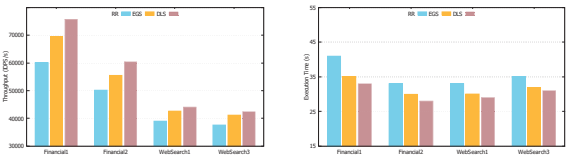
As shown in Fig. 5(a) and Fig. 5(b), when using RAID5 with 8 SSD, the IOPS and request execution time of the DLS scheduling strategy are improved more than 30% and 8.8% respectively compared with the RR and EGS scheduling strategy, similarly, for read-intensive traces, the improvement in IOPS and request execution time is small. This result is related not only to the optimization of the algorithm but also related to the increase in I/O performance as the number of SSD increases.

In summary, the DLS scheduling strategy performs better than the other two scheduling strategies for both read-intensive and a write-intensive trace.



(a) throughput (b) execution time

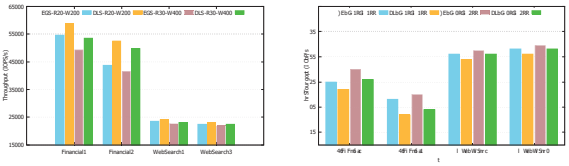
Fig. 4. RAID5 with 4 SSDs



(a) throughput (b) execution time

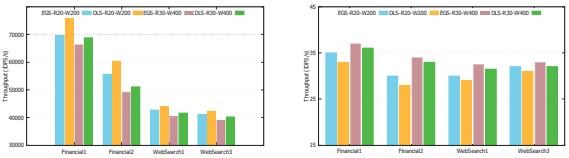
Fig. 5. RAID5 with 8 SSDs

2) Impact of page read and write latency on the scheduling strategies:



(a) throughput (b) execution time

Fig. 6. RAID5 with 4 SSDs



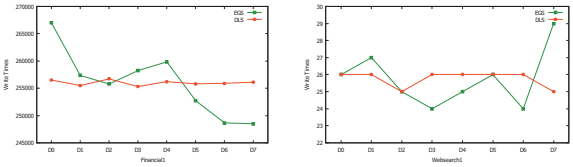
(a) throughput (b) execution time

Fig. 7. RAID5 with 8 SSDs

Read and write latency is an important factor affecting RAID5 performance. In this section, we compare the impact

of different read and write latency when using RAID5 with 4 SSD or 8 SSD. To evaluate the impact of page read and write latency on EGS and DLS, we set the read latency and write latency to be 20 us, 200 us, and 30 us, 400 us respectively. As shown in Fig. 6(a) and Fig. 6(b), when using RAID5 with 4 SSD, for write-intensive traces, the IOPS and request execution time of the DLS scheduling strategy is improved by 8.0% compared with EGS. As shown in Fig. 7(a) and Fig. 7(b), when using RAID5 with 8 SSD ,for write-intensive traces, the IOPS and request execution time of the DLS scheduling strategy is improved by 8.8% compared with EGS. It proves that our strategy not only has better processing effectiveness but also has higher stability. For write-intensive traces, the improvement is small whether for using RAID5 with 4 SSD or 8 SSD.

3) Write distribution comparison of different data traces: Each time the DLS processes a write request, it predicts the lifetime and delay of each SSD, and allocates write requests based on this result. Fig. 8(a) and Fig. 8(b) show the write distribution when using write-intensive trace and read-intensive trace, by recording the write times of each SSD, we can see that DLS has a better write balancing degree than EGS. This result shows that DLS is more efficient in IOPS and write balancing than other strategies.



(a) Financial1 (b) WebSearch1

Fig. 8. Write Distribution

V. Conclusion

In this paper, we propose a novel and efficient scheduling method named DLS for SSD-based array, which can improve the load balancing and lifetime of the array. DLS dynamically allocates the address of the next write request by predicting the latency of read and write requests and the lifetime of each SSD in the RAID5 array. To verify the effectiveness of the DLS strategy, we conduct a detailed evaluation and analysis to prove.The experimental results show that the I/O performance of DLS is 8% higher than that of EGS. In addition, the write distribution of DLS is more balanced than other strategies.

Acknowledgements

The work described in this paper is partially supported by the grants from the National Natural Science Foundation of China (61702357 and 61502309), Shenzhen Science and Technology Foundation (JCYJ20170816093943197 and JCYJ20170817100300603), Natural Science Foundation of Tianjin (18JCQNJC00300), Guangdong Natural Science Foundation (2016A030313045 and 2017B030314073), Guangdong Provincial General University National Development Program (2014GKXM054), Natural Science Foundation of SZU (803/0 00027060147 and 827-000073), and Research Grants Council of the Hong Kong Special Administrative Region, China (GRF 152223/15E, GRF 152736/16E and GRF 152066/17E), Peiyang Scholar Foundation of Tianjin University (2019XRG-0004).

References

[1] Y. Lu, C. Wu, and J. Li, "EGS: an effective global I/O scheduler to improve the load balancing of ssd-based RAID-5 arrays," in *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC), Guangzhou, China, December 12-15, 2017*, 2017, pp. 298–305.

[2] M. Bakratsas, P. Basaras, D. Katsaros, and L. Tassiulas, "Hadoop mapreduce performance on ssds for analyzing social networks," *Big Data Research*, 2017.

[3] J. G. Elerath and M. Pecht, "Enhanced reliability modeling of raid storage systems," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. IEEE, 2007, pp. 175–184.

[4] N. Jeremic, G. Mühl, A. Busse, and J. Richling, "The pitfalls of deploying solid-state drive raids," in *Of Systor: the Haifa Experimental Systems Conference*, 2011.

[5] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error characterization, mitigation, and recovery in flash-memory-based solid-state drives," *Proceedings of the IEEE*, vol. 105, no. 9, pp. 1666–1704, 2017.

[6] G. S. Choi and B.-W. On, "Study of the performance impact of a cache buffer in solid-state disks," *Microprocessors and Microsystems*, vol. 35, no. 3, pp. 359–369, 2011.

[7] Y. Li, P. P. C. Lee, and J. C. S. Lui, "Analysis of reliability dynamics of ssd raid," *IEEE Transactions on Computers*, vol. 65, no. 4, pp. 1131–1144, 2016.

[8] J. Guo, Y. Hu, and B. Mao, "Enhancing i/o scheduler performance by exploiting internal parallelism of ssds," in *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2015, pp. 118–130.

[9] B. Mao, S. Wu, and L. Duan, "Improving the ssd performance by exploiting request characteristics and internal parallelism," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[10] M. Jung and M. T. Kandemir, "Sprinkler: Maximizing resource utilization in many-chip solid state disks," in *IEEE International Symposium on High Performance Computer Architecture*, 2014.

[11] C. Min, K. Kim, H. Cho, S.-W. Lee, and Y. I. Eom, "Sfs: random write considered harmful in solid state drives." in *FAST*, vol. 12, 2012, pp. 1–16.

[12] L. M. Grupp, J. D. Davis, and S. Swanson, "The bleak future of nand flash memory," in *Proceedings of the 10th USENIX conference on File and Storage Technologies*. USENIX Association, 2012, pp. 2–2.

[13] L. Han, Z. Shen, Z. Shao, and T. Li, "Optimizing raid/ssd controllers with lifetime extension for flash-based ssd array," in *Proceedings of the 19th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*. ACM, 2018, pp. 44–54.

[14] H. Takahashi, K. Mori, and H. F. Ahmad, "Balanced memory architecture for high i/o intensive information services for autonomous decentralized system," in *2009 International Symposium on Autonomous Decentralized Systems*. IEEE, 2009, pp. 1–7.

[15] L. Xun and D. Li, "Pseudo-parallel i/o schedule of synchronous raid," in *International Conference on Computer Science & Software Engineering*, 2008.

[16] A. R. Abdurrab, X. Tao, and W. Wei, "Dloop: A flash translation layer exploiting plane-level parallelism," in *IEEE International Symposium on Parallel & Distributed Processing*, 2013.

[17] H. Yang, J. Hong, F. Dan, T. Lei, L. Hao, and R. Chao, "Exploring and exploiting the multilevel parallelism inside ssds for improved performance and endurance," *IEEE Transactions on Computers*, vol. 62, no. 6, pp. 1141–1155, 2013.

[18] U. T. Repository, http://traces.cs.umass.edu/index.php/Storage/Storage, 2007.