Balancing Spatial Locality with Parallelism in Solid State Disks

Myeong-Ho Lee¹, Jongmoo Choi² and Seungjae Baek^{3*}

¹Department of Ecommerce, Semyung University, Jecheon, Chungbuk, 27136, Korea; mhlee@semyung.ac.kr ²Department of Software Science, Dankook University, Yongin, Gyeonggi, 16890, Korea; choijm@dankook.ac.kr ³Korea Institute of Ocean Science and Technology, Ansan, Gyeonggi, 15627, Korea; baeksj@kiost.ac.kr

Abstract

Objectives: Modern flash memory based storage systems, such as Solid State Disks (SSDs), are actively utilizing the channel/way interleaving to exploit parallelism among multiple NAND chips. **Methods/Statistical Analysis**: However, the flip side of the interleaving is that it disperses data with spatial locality across different NAND blocks, which eventually causes a high garbage collection overhead. To overcome this problem, we propose a spatial locality-aware allocation policy, called SLAP. It uses the notion of stream, which is defined as a set of data having consecutive Logical Page Numbers (LPN). **Findings**: By allocating a stream into a NAND block separately, it can preserve the spatial locality. In addition, by handling multiple streams simultaneously, it can obtain the parallelism among NAND chips. Also, we discuss that SLAP can balance between locality-preserving and parallelism by providing a spectrum from a traditional parallelism-oriented allocation to a strict locality-preserving one. We have implemented SLAP on a page-level mapping Flash Translation Layer (FTL) that is being used as a default FTL in many commercial SSDs. **Improvements/Applications**: Trace-driven simulation based experimental results have shown that SLAP can improve performance by up to 35.3% with an average of 13.1%, compared with the traditional allocation policy for the three workload considered.

Keywords: Allocation Policy, Locality, NAND, Parallelism, SSDs

1. Introduction

SSDs are expanding their usage from laptop to high performance computing servers¹⁻³ and enterprise systems. Applications on these systems require new capabilities with various aspects such as capacity, performance, and reliability and so on. In order to meet these requirements, modern SSDs make use of several cutting-edge technologies.

Specifically, to enlarge capacity, SSDs adopt the multichannel multi-way architecture, consisting of multiple independent NAND chips. For instance, Intel X25M SSD consists of 10 channels and 2 ways per channel (total 20 chips)⁴, while Micron P420m SSD has 32 channels with4 ways per channel (total 128 chips)⁵. In addition, to improve performance, SSDs utilize write buffer and channel/way interleaving, which enable to exploit parallelism

*Author for correspondence

among multiple NAND chips⁶. To enhance reliability, advanced error correction code such as Low-Density Parity Check (LDPC)⁷, RAID-based scheme⁸ and wearunleveling technique⁹ are integrated into SSDs. Also, to increase applicability, SSDs employ new techniques such as de-duplication¹⁰ and caching mechanisms^{2,11}.

By the way, the channel/way interleaving is, in fact, a double-edged sword. On the positive side, it can boost performance by handling read/write requests in paralleling using multiple chips. However, the downside is that it disperses data with spatial locality across multiple chips. Data with spatial locality have a tendency to be updated or deleted at the same time. Hence, the dispersion make invalidated pages to be scattered across multiple blocks, which eventually deteriorates the garbage collection performance. To overcome this problem, we propose a novel allocation policy, called Spatial Locality-aware Allocation Policy (SLAP). The key idea of our proposal is introducing the concept of stream. This concept is devised to identify data that have a high possibility to be updated or deleted at the same time. Specifically, in this study, we define a stream as a set of write requests in the write buffers in SSDs, which have consecutive LPNs.

Once streams are identified, SLAP allocates them into different blocks in an isolated manner. This allocation enables data with spatial locality to be preserved in a block, allowing invalidated pages bounded in the block. Besides, by processing multiple streams simultaneously using multiple NAND chips, it still takes advantages of the channel/way interleaving effects. Also, we discuss that SLAP can balance between locality-preserving and parallelism by providing a spectrum from a traditional parallelism-oriented allocation to a strict locality-preserving one.

The proposed SLAP can be integrated into any type of FTLs. In this study, we have implemented SLAP on a page-level mapping FTL using the Disk¹² with SSD extension¹³. Experimental results have revealed that SLAP can improve performance by up to 35.3% with an average of 13.1%, compared with the traditional allocation policy for the three workloads considered.



Figure 1. General SSD architecture.

The rest of this paper is organized as follows. We describe related work in Section 2. Then, the key concept of our proposal is elaborated in Section 3. The existence of spectrum between parallelism and locality is discussed in Section 4. Section 5 shows the performance analysis results with Disk Sim. Finally, we give the conclusion and future work in Section 6.

Figure 1 shows the general architecture of SSDs. It consists of two main parts, SSD controller and NAND flash chips. The SSD controller consists of the host interface, processor, DRAM and flash controller. The processor and DRAM are used for executing FTL. Also, large portion of DRAM is used as the write buffer, a kind of caching area for keeping write requests. NAND flash chips are connected into the SSD controller through the flash controller, constructing multi-channel and multi-way structure. Hence, data on different chips can be accessed simultaneously using the channel/way interleaving.

Each NAND flash chip is divided in multiple blocks, which is further divided into multiple pages. A page is the unit for the read/write operations while a block is the unit for the erase operation. NAND flash memory has two unique characteristics, erase-before-write and a limited number of program/erase cycles. To deal with these characteristics, SSDs employ a software layer called FTL, which provides: 1. mapping for the out-place update, 2. garbage collection for reclaiming invalidated pages, and 3. wear-leveling for enhancing the lifetime of flash memories¹⁴.

Since FTL plays a key role for the performance and reliability of SSDs, it has been studied intensively during recent decades¹⁵⁻²⁰. Based on the mapping techniques, FTLs can be classified into three categories; page-level mapping, block-level mapping and hybrid mapping²¹. Page-level mapping FTLs supports high flexibility and performance while requiring large DRAM space to maintain mapping table with page granularity. On the contrary, block-level mapping FTLs can minimize the DRAM requirement while often suffering from performance degradation due to reclaiming. Hybrid mapping FTLs try to balance the memory requirement and performance¹⁵.

To reduce the memory requirement of the page-level mapping FTL, Gupta et al. propose DFTL (demand-based FTL)that keeps only recently referenced portion of the page-level mapping table in memory¹⁷. In²⁰ suggest Lazy FTL that updates the cached page-level mapping table in a lazy manner to enhance not only performance but also reliability²⁰. In¹⁹ designs Janus-FTL that finds optimal partitions where one partition is managed by page-level mapping while the other managed by block-level mapping¹⁹. Our proposal can be integrated into any type of FTLs. But, in this study, we mainly focus on the page-level mapping technique since it is commonly used in modern SSDs.

Some FTLs are closely related to our study in the aspect that they try to exploit locality. Jiang et al. propose Spatial locality aware FTL (S-FTL) that takes advantages of spatial locality to reduce the mapping table size and to increase hit ratio for in-cache mapping information¹⁸. Our approach is utilizing spatial locality to reduce the garbage collection overhead while supporting parallelism. Lee et al. design Locality-Aware Sector Translation (LAST) FTL that exploits sequential locality, a special case of spatial locality, for log buffer management and temporal locality for hot/cold segregation to reduce the garbage collection overhead¹⁶.However, they are focusing on the hybrid mapping schemes only and do not consider the parallelism issues. To the best of our knowledge, this study is the first one that attempts to preserve spatial locality, while supporting parallelism in SSDs.

2. Key Concept

In this section, we first discuss the problem of the traditional allocation policy used in SSDs using a walk-through example. Then, we elaborate how our proposal overcomes the problem. Figure 2 shows how the traditional policy allocates flash memory pages to service write requests.



(a) After programming write requests in a write buffer



(b) After deleting (or updating) data D7~D15

Figure 2. Traditional allocation policy specific time window or data with causality²².

In the Figure 2, we assume that SSD consists of four flash memory chips, each of which has four blocks, and each block has four pages. We also assume that there is no written data in the flash memory initially. Every request (Dn in this Figure 2, where n is a logical page number) is firstly stored in a write buffer and will be programmed into a page depending on FTL's allocation policy.

Figure 2a shows the contents of flash memory after programming all the write requests in the write buffer. For maximizing performance, the traditional allocation policy allocates pages in a round-robin manner (i.e., interleaving) to fully utilize parallelism. For example, D1 is stored on the first flash chip and D2 on the second flash chip, and so on.

After that, assume that D7~D15 are requested to be deleted (or updated) as shown in Figure 2b. Note that this sort of requests is frequently incurred on a file deletion or even for a very small update to a Word or a Power Point file as the file is compressed²¹. Invalidated pages exist all over the chips, causing the significant garbage collection overhead. Specifically, seven valid page copies for D1, D38, D2, D39, D22, D37 and D23 and four block erases are required.

With identical requests and configurations to Figure 2 and 3 shows how our proposed SLAP works. At first, SLAP tries to find streams in the write buffer. A stream can be identified from the write buffer in SSDs using various methods such as data with consecutive LPNs or data written within a in this paper, we mainly focus on the first method and define a stream as a set of write requests that have consecutive LPNs. Specifically; we introduce a control parameter called stream threshold. If the number of consecutive LPN sis larger than stream threshold, they are grouped forming a stream and allocated to a block. When SLAP identifies multiple streams, SLAP allocates them into different blocks in an isolated manner. In the example shown in Figure 3, we set the stream threshold as four. In Figure 3a, three streams are detected, namely D7~D10, D11~D14 and D22~D25, and allocated on different blocks while rest of the requests that are not a part of the detected streams are allocated like the traditional interleaved fashion.

Again, assume that D7~D15 are requested to be deleted. SLAP requires only three page copies and three blocks erases for a garbage collection, as shown in Figure 3b. It means that the proposed allocation policy enables

data with spatial locality to be preserved in a block, allowing invalidated pages bounded in the block.



(a) After programming write requests in a write buffer



(b) After deleting (or updating) data D7~D15 **Figure 3.** Spatial locality-aware allocation policy.

3. Intra-Stream Parallelism

In the case when there are no detected streams, SLAP distributes write requests using the conventional page allocation policy, interleaving them among multiple chips. Therefore, SLAP has no adverse effect on performance even with the completely randomized write requests. Note that, in actuality, SLAP can find out streams enough to utilize all NAND chips due to the fact that modern SSDs employ a large size write buffer, usually manipulating more than 4,000write requests^{22,24}. By processing multiple streams simultaneously, it takes advantages of the channel/way interleaving effects for write requests.

However, when we consider read requests, the story becomes different. For example, assume that D22~D25 are Again, requested for read in Figure 2b and 3b, respectively. In Figure 2b, the four pages can be read from four NAND chips in parallel while they are read from one chip in Figure 3b. It implies that SLAP has a potential to degrade the read performance when the requests had been handled as a stream and there are these read requests only in SSDs.

There are three feasible solutions to overcome this problem. One is a hardware-level approach, employing additional buffer in a NAND chip and exploiting a pipeline-like mechanism so that different components of the read latency of multiple requests could be overlapped in SSDs. Such mechanisms are actively studied in DRAM research area. However, it demands hardware modifications, which is beyond the scope of this paper. The second solution is applying data reorganization. In SSDs, during garbage collecting, we have a chance to rearrange data for various purposes such a shot-cold separation and static wear-leveling. At this time, we can recognize data that have been read intensively in a read-only manner and redistribute them across multiple blocks. The third solution is utilizing the parallelism in a stream. In this paper, we concentrate the third one, leaving the second one as the future work.

The control parameter, *stream_threshold*, can be set at the ranges from 1 to 'pages per block' (PPB). When it is set as1, SLAP behaves like the traditional policy, distributing all requests in an interleaved manner. At this point, we can make full use of parallelism. When the parameter becomes larger, SLAP tries to detect streams and allocates each block into different blocks to obtain the locality preserving benefit. In other word, by controlling the parameter, SLAP supports a spectrum between parallelism and locality, providing an optimal value by balancing these two aspects.

To enhance the read performance, we devise a mechanism that divides a stream further for exploiting the intra-stream parallelism. Specifically, we introduce a new control parameter, called Δ , and partition a stream into Δ sub-streams. Each sub-stream is allocated into a different block. The value of Δ is determined by the ratio between the read latency of a NAND chip and the sum of other FTL overheads (for short, we refer to it as a FTL overhead). For instance, when we assume the read latency as 150 us and the FTL overhead as 50 us, Δ becomes 3, as illustrated in Figure 4. As summary, the stream threshold is devised to balance the parallelism and locality for write requests, while Δ to control the degree of parallelism for read requests.



Figure 4. Δ for overlapping the read latency and FTL overhead while preserving locality.

Now we would like to mention about the tradeoff between performance and energy consumption. By uti-

lizing multiple channel, way and NAND chips, we can obtain better performance. However, in terms of energy consumption, providing relevant performance only with small number of channel/way is very desired²³. It means that SLAP efficiently controls not only the tradeoff between spatial locality and parallelism in terms of performance but also the tradeoff between power consumption and performance.

4. Performance Evaluation

To quantitatively evaluate the effectiveness of SLAP, we have built two synthetic workloads and have chosen three real worlds workloads²⁵⁻²⁷. We have implemented SLAP at Disk Sim 4.0 with SSD extension^{12.13}. We set the page size as 4 KB and PPB as 128, and we use default values for all other parameters. Each experiment was run on the simulator five times, and we use an average value of the executions when reporting results. Before each measurement, the simulated storage space is fully filled with the workloads for each of those experiments.

4.1 Synthetic Workload

Figure 5 shows the results from the synthetic workloads. Specifically, Figure 5a and 5c present I/O Per Second (IOPS) for both of sequential and random workloads when we vary stream threshold from 8 to 128 while Δ is set as1. Figure 5b and Figure 5d present how many streams are identified during the tests. Observations from the results can be summarized as follows:

Depending on the characteristics of each workload, optimal stream threshold is varied, and the number of identified stream is the main driving factor in performance. Specifically, SLAP enhances performance up to 325.9% (134.2% on average) for the sequential workload, and up to 36.4% (1.84% on average) for the random workload. These performance gains are due to the reduction of the number of copies and erases, as shown in Table 1. Even when no stream is detected, there is no hazard at all because SLAP allocates pages in a conventional manner in that case. Therefore, as expected, there are no differences when no stream is found.

The optimal value of stream threshold is 16 and 8, for sequential and random workload, respectively. When it becomes larger, the number of pages included in streams



(a) Sequential: IOPS



(b) Sequential: stream page identification







(d) Random: stream page identification **Figure 5.** Synthetic workload.

Items	Stream_ threshold	# of erase		# of copy	
		w/ SLAP	w/o SLAP	w/ SLAP	w/o SLAP
Seq.	8	45,874	47,082	14,011	166,334
	16	45,792	47,082	4,084	166,334
	32	46,488	47,082	92,055	166,334
	64	46,142	47,082	128,373	166,334
	128	47,082	47,082	166,334	166,334
Rand.	8	35,126	35,304	27,060	49,492
	16	35,266	35,304	44,925	49,492
	32	35,292	35,304	47,938	49,492
	64	35,292	35,304	47,938	49,492
	128	35,292	35,304	47,938	49,492
Proxy	8	115,486	115,804	0	39,615
	16	115,482	115,804	0	39,615
	32	115,490	115,804	0	39,615
	64	115,486	115,804	0	39,615
	128	115,488	115,804	0	39,615
OLTP	8	10,077	10,433	436,572	442,070
	16	10,400	10,433	441,000	442,070
	32	10,400	10,433	441,000	442,070
	64	10,400	10,433	441,000	442,070
	128	10,400	10,433	441,000	442,070
Concat	8	538,106	537,018	4,040,505	3,901,669
	16	536,984	537,018	3,898,198	3,901,669
	32	535,370	537,018	3,692,689	3,901,669
	64	535,512	537,018	3,711,296	3,901,669
	128	535,670	537,018	3,730,982	3,901,669

Table 1. Number of erase and copy.

decreases, leading to less performance improvement. When it becomes smaller than the optimal value, the performance gains become smaller even though SLAP detects more streams, because several streams are interweaved in a block, resulting in the increasing of the number of copies and erases. It implies that stream threshold needs to be set enough to preserve the spatial locality in an isolated manner.

 When we change Δ as 2 or 4, the performance gains show similar trends when we set a smaller stream threshold. For instance, performance measured under the configuration of stream threshold as 16 and Δ as 2 is similar to that observed under the different configuration when stream threshold is 8 and Δ is 1. The reason is that, in two configurations, the number of pages written in a block is the same, that is 8, and the out-of-order executions in SSDs allows read requests to be served in a similar fashion. Besides, these workloads issue requests enough to make most of NAND chips busy.

4.2 Realistic Workloads

Let us look at the effectiveness of SLAP with well-known realistic workloads. Figure 6 shows IOPS and stream page identification results for Proxy, OLTP, and Concat workloads.



(a) Proxy: IOPS



(b) OLTP: IOPS



(c) Concat: IOPS



(d) Proxy: stream page identification



(e) OLPT: stream page identification



(f) Concat: stream page identification **Figure 6.** Realistic workload.

- For Proxy, a large amount of streams, which is sufficient to improve the performance, are identified. Specifically, performance improvement with SLAP was (very consistently) 35.3% on the average. With SLAP, the number of copies, which is the main source for this improvement, is zero as shown in Table 1. It implies that SLAP can differentiate invalid data that are deleted/ updated together from valid data.
- In OLTP, only small numbers of streams are detected because it primarily has very random small size requests. However, as we already have seen, there's no harm in performance, and what is more, those small numbers of streams facilitate up to 7.2% and 2.1% on average performance improvements. It also uncovers that random workloads prefer smaller stream threshold as discussed in Figure 5c.
- The Concat workload shows very interesting results. Even though there are a considerable number of streams (up to 38.2% and 12.7% on

average), the performance improvement is 4.7% at maximum and only 2.0% on average. The biggest performance gain is obtained when stream threshold is 32. When it is 8, SLAP degrades performance due to the increment of copies, as shown in Table 1. Our closer investigation shows that some requests which are never accessed again are identified as streams, and reside in a block, deteriorating the garbage collection overhead. This problem is caused because smaller stream threshold makes streams intermixed in a block and some are modified/ deleted while others are not. We can overcome this problem by segregating new writes and writes from the garbage collection into different blocks.

5. Conclusion

This paper proposes a novel spatial locality-aware allocation policy called SLAP. We carefully argue that utilizing interleaving without considering the locality makes storage performance worse over time, especially in SSDs. It can provide high performance at the initial stage, but the performance drops significantly as time goes and utilization becomes higher at the steady state. Our experiments shows that SLAP preserves data with locality in a block in SSDs while exploiting parallelism, resulting in performance improvements compared with the traditional allocation policy.

We envision a couple of directions for future research. First, we will explore the dynamic stream threshold adjusting. It is a relevant approach as workloads are continuously changed in everyday SSDs. Second, we will extend SLAP to enhance the parallelism for read requests including the hardware acceleration and data reorganization. Also, we investigate the effect of intra-stream parallelism with more read-intensive workloads.

6. References

- 1. Kim Y, Oral S, Shipman G, Lee J, Dillow D, Wang F. Harmonia: A Globally Coordinated Garbage Collector for Arrays of Solid-State Drives, In: Proceeding of the IEEE 27th Symposium on Mass Storage Systems and Technologies, Denever, CO, 2011 May, 1–12.
- 2. Albrecht C, Merchant A, Stokely M, Waliji M, Labelle F, Coehlo N, Shi X, Schrock CE. Janus: Optimal Flash

Provisioning for Cloud Storage Workloads, In: Proceedings of the USENIX Annual Technical Conference, Berkeley; 2013. p. 91–102.

- Monfared V, Hassan M, Daneshmand S, Taheran F, Kaewtrakulpong P. Effects of Geometric Factors and Material Properties on Stress behavior in Rotating Disk, Indian Journal of Science and Technology. 2014; 7(1):1–6.
- Yoo B, Won Y, Choi J, Yoon S, Cho S, Kang S. SSD Characterization: From Energy Consumption's Perspective, In: Proceedings of the 3rd USENIX Conference on Hot Topics in Storage and File Systems, Berkeley, CA, USA, 2011, 3–3.
- MICRON. P420m Architecture. Date Accessed: 2016. Available at: http://www.micron.com/products/solid-statestorage.
- Agrawal N, Prabhakaran V, Wobber T, Davis JD, Manasse M, Panigrahy R. Design Tradeoffs for SSD Performance, In: Proceedings of the USENIX Annual Technical Conference, Berkeley, CA, USA, 2008, 57–70.
- Zhao K, Zhao W, Sun H, Zhang T, Zhang X, Zheng N. LDPC-in-SSD: Making advanced Error Correction Codes Work Effectively in Solid State Drives, In: Proceedings of the 11th USENIX Conference on File and Storage Technologies; 2013. p. 243–56.
- Kim J, Lee J, Choi J, Lee D, Noh S. Improving SSD Reliability with RAID via Elastic Striping and Anywhere Parity, In: Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Budapest; 2013. p. 1–12.
- Jimenez X, Novo D, Ienne P. Wear Unleveling: Improving NAND Flash Lifetime by Balancing Page Endurance, In: Proceedings of the 12th USENIX Conference on File and Storage Technologies; 2014. p. 47–59.
- Chen F, Luo L, Zhang X. CAFTL: A Content-Aware Flash Translation Layer Enhancing the Lifespan of Flash Memory Based Solid State Drives, In: Proceedings of the 9th USENIX Conference on File and Storage Technologies; 2011. p. 6–6.
- 11. Oh Y, Choi J, Lee D, Noh SH. Caching Less for Better Performance: Balancing Cache Size and Update Cost of Flash Memory Cache in Hybrid Storage Systems, In: Proceedings of the 10th USENIX Conference on File and Storage Technologies; 2012. p. 25.
- Bucy JS, Schindler J, Schlosser SW, Ganger GR. The DiskSim Simulation Environment Version 4.0 Reference Manual, Technical report, CMU-PDL-08-101, Carnegie Mellon University, 2008.
- 13. SSD Extension for DiskSim Simulation Environment. Date Accessed: 06/03/2009. Available at: http://research.microsoft.com/en-us/downloads.
- 14. Gal E, Toledo S. Algorithms and Data Structures for Flash Memories, ACM Computing Surveys. 2005; 37(2):138–63.

- Kim J, Kim JM, Noh S, Min SL, Cho Y. A Space-Efficient Flash Translation Layer for Compact Flash Systems, IEEE Transactions on Consumer Electronics. 2002; 48(2):366–75.
- Lee S, Shin D, Kim YJ, Kim J. LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems, SIGOPS Operating System Review. 2008; 42(6):36–42.
- Gupta A, Kim Y, Urgaonkar B. DFTL: A Flash Translation Layer Employing Demand-Based Selective Caching of Page-Level Address Mappings, In: Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems; 2009. p. 229–40.
- Jiang S, Zhang L, Yuan X, Hu H, Chen Y. S-FTL: An Efficient Address Translation for Flash Memory by Exploiting Spatial Locality, In: Proceedings of the IEEE 27th Symposium on Mass Storage Systems and Technologies, Denver, CO, 2011, 1–12.
- Kwon H, Kim E, Choi J, Lee D, Noh SH. Janus-FTL: Finding the Optimal Point on the Spectrum Between Page and Block Mapping Schemes, In: Proceedings of the 10th ACM International Conference on Embedded Software, Scottsdale, Arizona, USA, 2010, 169–78.
- 20. Ma D, Feng J, Li G. Lazy FTL: A Page-Level Flash Translation Layer Optimized for NAND Flash Memory, In: Proceedings of the ACM SIGMOD International Conference on Management of Data, Athens, Greece, 2011, 1–12.
- 21. Yoo J, Won Y, Hwang J, Kang S, Choi J, Yoon S, Cha J. VSSIM: Virtual Machine Based SSD Simulator, In: Proceedings of

the IEEE 29th Symposium on Mass Storage Systems and Technologies, Long Beach, CA, 2013, 1–14.

- 22. Kim J, Lee C, Lee S, Son I, Choi J, Yoon S, Lee HU, Kang S, Won Y, Cha J. Deduplication in SSDs: Model and quantitative analysis, In: Proceedings of the IEEE 28th Symposium on Mass Storage Systems and Technologies, San Diego, CA, 2012, 1–12.
- 23. Muniswamy-Reddy KK, Holland DA. Causality-Based Versioning, In: Proceedings of the 7th Conference on File and Storage Technologies; 2009. p. 15–28.
- 24. Kim H, Ahn S. BPLRU: A Buffer Management Scheme for Improving Random writes in Flash Storage, In: Proceedings of the 6th USENIX Conference on File and Storage Technologies; 2008. p. 1–31.
- Megiddo N, Modha DS. ARC: A Self-Tuning, Low Overhead Replacement Cache, In: Proceedings of the 2nd USENIX Conference on File and Storage Technologies; 2003. p. 115–30.
- 26. Narayanan D, Donnelly A, Rowstron A, Donnelly A, Rowstron A. Write off-loading: Practical Power Management for Enterprise Storage, In: Proceedings of the USENIX Conference on File and Storage Technologies; 2008. p. 253–67.
- 27. ONeil EJ, ONeil PE, Weikum G. The LRU-K Page Replacement Algorithm for Database Disk Buffering, In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, 1993, 297–306.