Research Problems and Opportunities about Flash Memory *

Yichen Jia
Computer Science & Engineering
Louisiana State University

Abstract

Flash memory has been popular in recent years, employed in server, desktop and mobile environment. Its virtues such as high-speed, persistence and acceptable price have gained lot of attention from the research community. Compared with DRAM-based memory, flash memory is more scalable and energy efficient. Meanwhile, compared with hard disk drive (HDD), flash memory has high-speed, which becomes increasingly important for modern computer systems. However, its drawbacks such as endurance problem, reliability issues and random writes bottleneck are also the serious problems that need to be addressed properly. So lots of research opportunity can be found on flash memory.

In this article, we describe some promising research problems at both hardware and software level. Also, attempts that try to provide a more efficient communication scheme between hardware and software are also explored. After discussing these research papers, we propose some research directions and opportunities that are promising about flash memory.

1 Introduction

Memory and Storage subsystems are critical components of all the modern computer systems. However, recently, both memory and storage systems are exposing several serious problems. Firstly, DRAM-based memory is expensive and energy-hungry relatively [23]. As a result, when the data explodes, it is hard to scale accordingly. Secondly, performance gap between memory and storage device becomes increasingly large. Storage device has become a bottle as lot of applications become data intensive [3].

Fortunately, flash memory provides an opportunity to rethink these problems. On one hand, flash memory is affordable and energy efficient compared with DRAM-based memory. So by designing new DRAM/Storage architecture with flash memory, it is possible to address the scalability problem. On another hand, flash-based solid state drive (SSD) may serve as a storage device because it is persistent. Actually, SSD has been widely employed in desktops and servers. So it is worthwhile to look into the flash memory and flash-based SSD to address the corresponding problem about it.

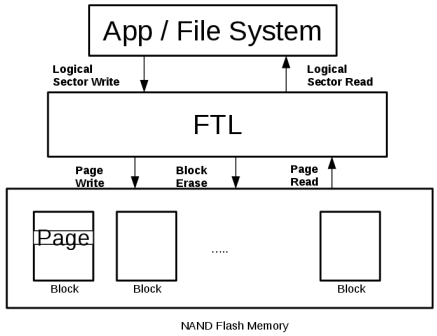

Figure 1: This figure demonstrates that the IO stack from application or file system to the device medium. The optimization opportunity can be found on the file system, FTL and physical medium. The information can be passed down to the FTL level, and also FTL can also be leveraged to file system.

As the Figure 1 shows, the optimization opportunity can exist at the hardware level, software level and their communication mechanisms.

On the *hardware front*, a comprehensive and powerful FTL resides in the flash-based SSD. It has both pros and cons. On one hand, the page mapping, wear leveling, garbage collection and other processes can be hidden from host, which provides an easy interface compatible to HDD. However, on the other hand, the management scheme inside the

*CSC 7999 Reading Report, Fall 2016

device is so complex that it is also hard to scale as the device size increases rapidly [2], because of the production cost or resource limitation inside it.

On the *software front*, it is worth exploring how to use flash memory efficiently by the application. So far, applications, such as caching, key-value store, database, file system, have been designed for the underlying flash memory or flash-based SSD. We think there are still other proper applications can be designed or optimized.

On the *inter-layer communication front*, three directions can be found. First, to provide an enriched interface between hardware and software. Second, the software can manage the flash directly by removing the FTL. Finally, to move the flash management schemes to the host side to make use of resources on host. Researchers need to find the proper design approach when dealing different scenarios.

The paper is organized as follows: Section 2 summarizes the research work at the hardware level. Section 3 is about the work at the software level. Section 4 discusses about the papers that focus on the hardware and software interface. Section 5 is the research opportunity that exposes from these papers. Section 6 concludes this paper.

2 Hardware Level Research

2.1 Properties of Flash Memory

There are two types of flash memory: NOR and NAND. They are different in their architecture and design. NOR flash uses no shared components, which allow random data access. NAND flash uses more compact technology to provide greater density, which behaves better in serial operation [37]. Relatively, NAND flash is more popular now because both the individuals and enterprises need larger devices in the "big data" era. There are five types of NAND flash: single-level cell(SLC), multiple-level cell(MLC), enterprise MLC(eMLC), Triple-level cell(TLC) and Vertical/3D NAND. Among them, SLC is the fastest and most durable. But it has smallest density compared with other types of NAND flash. Compared with SLC flash, MLC flash has a larger density, but it is slower and less durable than SLC [37].

Flash memory has four unique characteristics. (1) *High-speed.* Typically, flash memory reads data at the granularity of flash page and the latency of reading a Page (4K) from the media to the data register is about 25 μs [1]. As for the write operation, it takes about 200 μs from register to physical medium and the erase operation takes about 1.5 *ms*. (2) *Limited erase times.* The lifetime of flash memory is limited by the finite number of erase operations of each block before retirement [1] [34]. (3) *Less reliable than HDD.* Data errors caused by read/write disturbing and retention errors are more common on the flash memory than hard disk drive[34]. (4) *No in-place update.*The whole flash block needs to be erased before writing the new data [1] [34].

2.2 Performance of Flash-based SSD

There are four factors that affect the performance of flash-based SSD. Respectively, they are error correction code(ECC), over-provisioning space(OPS), garbage collection process(GC) and workload characteristics. To improve the reliability of flash memory, an ECC mechanism is deployed to reduce the the number of bit errors reported to the application and extend the lifetime of flash memory. Over-provisioning space is a part of physical space that is reserved for background operations, but it is not visible to the users. Garbage collection process is triggered when the number of free blocks is under some threshold. As flash memory doesn't support the in-place update, the valid page in some free blocks must be copied and rewritten to other free or partially free blocks, then the full block is erased. Finally, the workloads also have impact on the SSD performance, since sequential write throughput of SSD is good but random write throughput is not so good relatively.

ECC: FlexECC [14] uses the information passed down by upper-layer applications to distinguish the importance of each block, and chooses to apply the regular ECC or light weight EDC on each block in the cache scenario to reduce the computation overhead. They show that the computation overhead of ECC is quite worth noticing. IR [32] applies weak ECC at the early stage of SSD and incrementally reinforces ECC capability as the SSD wears out. This design is based on the fact that the probability that retention and programming errors happen

will improve as the programming times increase. It will reduce the the computation overhead of ECC by choosing proper ECC capability at different stages.

Garbage Collection: Harmonia [19] proposes to coordinate the garbage collection process in the SSD RAID because the garbage collection process will be the bottleneck of the whole system. By a global coordination, SSD RAID can provide a better response time and more predictable performance.

Over-provisioning: SDF [31] tries to eliminate the over-provisioning space by exposing each channel to the client as a device. They show that this design can effectively organize the data and improve the performance because SDF can access the raw flash device directly. Shen *et al.* [35] optimize the flash-based key value cache by reducing the over-provisioning space with cache system control controlling the open-channel SSD directly. They call this problem over-overprovisioning in their paper. Meanwhile they found out the double garbage collection problem and double mapping problem. Namely, the garbage collection happens both at the key-value cache system and in the device. Meanwhile, there is mapping from key to (slab,slot) pair in the application and there is another mapping from logical address to physical address in the device.

Random Writes: BPLRU [18] uses a buffer to absorb the random write and combine them into a sequential write with a block padding the least recently used method at the device level.

2.3 Reliability of Flash Memory

There are five main factors that affect the reliability of flash-based SSD, which are SLC vs. MLC NAND flash technology, incoming write traffic, PE cycles, write amplification and over-provisioning space respectively. The SanDisk report [34] mentions that MLC flash memory allows to store multiple bits of information such that its density is greater. However, the reliability decreases as the capacity increases. Also, the more write traffic comes into the device, the more worn out the device will be [5]. PE cycles are directly related to the reliability of flash memory, as the more times the flash memory is programmed and erased, the less reliable it will be [1] [34]. What's more, write amplification is a contributor to the device aging. The larger the write amplification factor is, the faster the device will become less reliable [34]. Finally, over-provisioning space is important for both the endurance and the performance. The larger the OPS is, the better the endurance and performance are [1] [34]. Research work about the reliability issue of flash memory mainly focuses on two fields: measurement and endurance improvement respectively. We will talk about them in details.

Measurement Tseng *et al.* [38] conduct a study on the flash memory chip level. They found out that the error rate does decrease when interrupted closer to completion. Also a power failure during one uncompleted operation will cause error to the previous completed operation. Zheng *et al.* [41] make a measurement on the device level, which showed that the simplest chip-level failure may be hidden at the device level. They revealed that bit corruption, shorn writes, unserializable writes, metadata corruption and dead device can be found at the device level when power failure happened suddenly. Laura *et al.* [10] point out that based on their analysis on many flash chips, the performance and reliability degradation will be significant as the density increases in the SSD in the future. So the scalability problem may be exposed in the future soon according to their result.

Endurance Improvement Liu *et al.* [26] propose that the retention capability of SSD is typically of years and most data is overwritten in hours or days. So this gap can be explored to improve the performance or extend the lifetime of the device. WARM [11] selectively relaxes the retention requirements by recognizing the write-hotness data in the device, because these data will be overwritten soon. In this way, they can prolong the lifetime of the SSD device greatly. Huang *et al* [15] use the retired blocks to store the data that only requires short retention time. Thus the device lifetime can be significantly extended because the retired block can always serve until it reaches some threshold. DeltaFTL [39] proposes to store the difference between the new version data and the old version data instead of all the new data to reduce the size of write operation and the times of garbage collection, which can improve the endurance. Aviad *et al.* [42] claim that compression within the SSD is better in performance than compression in file sys-

tem and application. CAFTL [5] extends the lifetime of the flash-based SSD device by reducing the incoming traffic with deduplication technique with minimum performance overhead by implementing several acceleration methods. It shows that deduplication at the device level is also feasible.

3 Software Level Research

Even though flash-based SSD can provide the interface compatible to HDD, how to make use of the flash efficiently still needs to be considered. Here we summarize three kinds of typical applications that fully explore the high-performance read merits and overcome the endurance and poor random writes performance shortcomings. They are caching, file systems & databases and key-value stores respectively. Since SSD can provide high-speed read operation and the price is lower than the DRAM-based memory, it is reasonable to put it between DRAM-based memory and HDD to serve as the cache for the HDD. However, this architecture will suffer heavy write traffic problem and other problems in the cache, which needs more attention to address. What's more, since flash-based SSD shows different performance and reliability characteristics with HDD, the original file systems and databases that are designed for HDD should be reconsidered and optimized for SSD specifically. Memory-based key-value store is popular recently, however, it cannot scale easily considering the price of DRAM and it energy cost. As a result, flashed-based key-value store or key-value cache is proposed to support the Internet based service. Here we discuss them in details.

3.1 Caching

Qin *et al.* [33] design two reliable writeback methods using write barriers, which are called write-back flush and write-back persist to handle different failure scenarios. They propose that the data in the same application can be flushed if needed. Oh *et al.* [30] balance the hit ratio in the SSD cache and the update cost for optimal performance, because increasing over-provisioning space will reduce the hit ratio but reduce the update cost. The tradeoff between the hit ratio and device response time can help reach the optimal performance for different workloads. CacheDedup [24] integrates caching and deduplication into one layer and proposes two replacement algorithms that are designed for deduplication scenario. Deduplication at application level can help enlarge the cache size and improve the device response time as less data written into the device. Soundararajan *et al.* [36] propose to use hard disk drive as a write cache for solid state drive because hard disk drive can provide comparative sequential write bandwidth than solid state drive and overwrites often happen in the cache scenario. HDD doesn't have that serious endurance problem that exists in SSD, so it fits to serve SSD as the write cache. Feng and Schindler [9] explore the deduplication effectiveness in large SSD cache inside VM hypervisors, which is different from others as it serves to different workloads at the same time.

3.2 File Systems & Databases

Canim *et al.* [4] present an SSD caching system for database located between DRAM and hard disk drive. They use a temperature monitor to determine whether the data will be replaced. The cold data cannot evict the warm data. This method uses the statistic information to help to make the replacement decision, which reduces the cache pollution problem. In this way, the disk I/O is reduced and the performance gain is obvious for the random IO. SFS [29] improves the bandwidth by writing sequentially in a log structure and grouping data together on writing by putting data block that have similar likelihood together. Since the performance of sequential write is much better than random write, SFS can improve writing efficiency greatly. What's more, random write is considered harmful for SSD, SFS can improve the device endurance as well. Finally, since data with similar update probability has been grouped together, the cleaning efficiency is also improved. F2FS [20] is designed to specifically for flash storage which uses the flash-friendly on-disk layout and cost-efficient index structure. F2FS aligns the data structure with the FTL operation unit and cleans using multi-head logs and hot/cold data separation. It's worth mentioning that F2FS avoids the metadata update propagation by using indirec-

tion for inodes and pointer blocks. To achieve a high utilization, F2FS adopts an adaptive writing policy and switches to threaded logging at the right time. Also, the performance will not degrade significantly even at a high utilization rate.

3.3 Key-Value Stores

FlashStore [7] designs a key value flash cache for back-end store such as MySQL, which organizes data in log structure and index data using hash table. Each look-up is one page read on flash. FlashStore uses a write buffer to accumulate writes so that every write to the flash page has sufficient data in it. Also, it keeps the recency bit vector to provide the recency information to help the key-value pair eviction. Besides a disk-presence bloom filter is added to reduce the unnecessary disk read latency. SILT [25] builds a high-performance flash-based key-value store by combining three kinds of popular key-value stores in a memory-efficient way. Sorted store is used to keep the most memory-efficient index. LogStore is to serve the the keys insertion because it is write-friendly. HashStore is added to keep the worst case performance. This kind of multi-store design can optimize from different aspects. NVMKV [28] puts little data management strategies on the key-value store, instead, it makes use of the complicated schemes that are becoming available in the FTL layer to provide the high-performance speed. Many functions in modern FTLs such as log-structure, dynamic data mapping, hashing and so on are quite similar to key value store functions. So to make use of these advanced function by system or application is promising. NVMKV achieves good performance behavior by using these FTL primitives. SkimpyStash [8] provides a high-throughput flash-based key-value store that needs extremely low memory footprint at about 1 byte per key-pair pair. Their effort is mainly on removing the flash pointer from memory to flash to save the memory overhead.

4 Hardware/Software Interface

Researches that co-design the hardware and software are quite popular and promising. They may have three main ways to achieve this purpose: (1) Enrich FTL interface. (2) Remove the FTL. (3) Leverage FTL to Host.

All of these three approaches have their pros and cons. A wider FTL interface can obtain more semantic information from the system or application level. However, it also needs more hardware resources in the device and complicated management schemes which may affect the reliability of the device. By removing the FTL, the application can access the high-speed raw device. However, physical medium details of different vendors and different models may be an obstacle for the application to manage. Also, without the protection of FTL, raw flash is less reliable than the SSD, so the reliability issue may arise in this case. Finally, by moving the functionality in FTL to host, it can use much more host-side hardware resources and semantic information to allocate the metadata and data and guarantee the data integrity intelligently. However, it depends on whether this approach is feasible for this application. This approach is not generic because each application may need a specific design considering its own behavior and requirement.

4.1 Enriched FTL Interfaces

Because FTL doesn't support the in-place update, JFTL [6] remaps the address of journaling file system changes to the home address of the change for the purpose of reducing the update traffic. Meanwhile, JFTL can guarantee the consistency of the journaling file system even though it eliminates lots of redundant writes. JIT-GC [12] triggers the garbage collection process only when it is necessary considering the incoming traffic in the future. They find that the time when the garbage collection process is triggered is important for GC efficiency. Meanwhile, JIT-GC reserves large over-provisioning space to keep the high-performance of writes. Multi-Streamed SSD [17] requires the host to send the SSD device the data lifetime such that the semantic gap between host and device is reduced. Then the device can use the information from the host to deal with the similar stream together and place the data with similar lifetime together. Also, different streams will be put on different locations so that they will not disturb each other. Nameless Writes [40] provides a new device

interface to enable the device to control the block allocation decision. In this case FTL can exploit rich file system information to optimize the data management inside.

4.2 FTL Removal

NoFTL [13] proposes to integrate some of the FTL functionality into the database system to simplify the IO stack and improve the raw device access speed. Meanwhile, device driver also helps collect the access information to help the application to optimize its data allocation. They show that this optimization can improve the performance to a great extent. Yaffs [27] also tries to handle the raw flash device to get the high-speed of flash memory through the NAND flash interface with a log structure. As there is no FTL, the memory footprint and CPU overhead is small relatively. As the raw flash device is faster, the performance of Yaffs is good. However, the reliability issue must be addressed well at file system level.

4.3 Host-managed Flash

DFS [16] leverages the FTL layer to the driver layer to manage the raw flash device to make use of more resource and information at the host side. But some problems such as double garbage collection at FTL and some applications still remains. SDF [31] tries to expose each channel on the device as an individual device and eliminates the over-provisioning space. What's more, it can explore the internal parallelism to the maximum and provide a more flexible interface. This hardware/software co-design gains great performance benefit. But this design is not generic [22]. Application-managed Flash [22] proposes a new block interface that doesn't permit overwriting. They show that applications can deal with this restriction well by append-only segments. Also, it doesn't expose the physical details, making it more portable and generic. REDO [21] removes the garbage collection and the-logical-to-physical mapping in the FTL and merge them with the LFS into refactored file system (RFS). So RFS can manage the physical blocks directly and the double garbage collection processes are reduced to one. In the device, only bad block management scheme and wear leveling scheme and other simply techniques are kept.

5 Research Opportunity

Based on the papers listed above, there are at least five promising research fields, which include: (1) how to make use of the high-performance and parallelism characteristic of the raw flash memory to provide the high-throughput and low-latency service. (2) To use the system-level or application-level information to help to explore the semantic information at the FTL level. (3) To leverage the FTL to the upper layer to make full use of the host resource to efficiently manage the device. (4) To use the software method to extend the lifetime of SSD such as deduplication and compression. (5) To relax the persistence requirement for performance based on the lifetime of the data.

6 Conclusions

Flash memory has attracted lots of research interest in recent years from hardware level to software level, as well as at the hardware/software interface. Some research papers try to make use of the merits of flash memory such as high-performance, acceptable price, persistence and so on. Others try to overcome the limitation of flash memory such as endurance problem, reliability issues and random write bottleneck. The solutions on the hardware level may know more physical details and can provide a simpler interface to system or applications. On the contrary, the solutions on the software level try to make use of the flash-based SSD like a black box, but still, optimization approaches are quite helpful to improve the performance and extend the lifetime of SSD. As for those work on the interface, they all have their advantages and limitations. There is not a general purpose design that fits all the scenarios. It depends on the application or system requirements to determine which approach to choose.

References

[1] AGRAWAL, N., PRABHAKARAN, V., WOBBER, T., DAVIS, J. D., MANASSE, M., AND PANIGRAHY, R. Design tradeoffs

for ssd performance. In *Proceedings of ATC'08* (Boston,MA, Jun 2008).

[2] AMAZON. 1tb ssd is avaiable on market. https://www.amazon.com/Samsung-2-5-Inch-SATA-Internal-MZ-7TE1T0BW/dp/B00E3W16OU.

[3] BRYANT, R. E. Data-intensive supercomputing: The case for disc. *CMU CS Tech. Report* (2007), 07–128.

[4] CANIM, M., MIHAILA, G. A., BHATTACHARJEEA, B., ROSS, K. A., AND LANG, C. A. Ssd bufferpool extensions for database systems. In *Proceedings of VLDB'10* (Singapore, Sept 2010).

[5] CHEN, F., LUO, T., AND ZHANG, X. CAFTL: a content-aware lash translation layer enhancing the lifespan of flash memory based solid state drives. In *Proceedings of FAST'11* (San Jose,CA, Feb 2011).

[6] CHOI, H. J., LIM, S.-H., AND PARK, K. H. JFTL: a flash translation layer based on a journal remapping for flash memory. *ACM Transactions on Storage 4*, 4 (Jan 2009).

[7] DEBNATH, B., SENGUPTA, S., AND LI, J. Flashstore: High throuphput persistent key-value store. In *Proceedings of VLDB'10* (Singapore, Sept 2010).

[8] DEBNATH, B., SENGUPTA, S., AND LI, J. Skimpystash: Ram space skimpy key-value store on flash-based storage. In *Proceedings of SIGMOD'11* (Athens, Greece, Jun 2011).

[9] FENG, J., AND SCHINDLER, J. A deduplication study for host-side caches in virtualized data center environments. In *Proceedings of MSST'13* (Long Beach, CA, May 2013).

[10] GRUPP, L. M., DAVIS, J. D., AND SWANSON, S. The bleak future of nand flash memory. In *Proceedings of FAST'12* (San Jose, CA, Feb 2012).

[11] GRUPP, L. M., DAVIS, J. D., AND SWANSON, S. WARM: improving nand flash memory lifetime with write-hotness aware retention management. In *Proceedings of MSST'15* (Santa Clara,CA, May 2015).

[12] HAHN, S. S., LEE, S., AND KIM, J. To collect or not to collect: Just-in-time garbage collection for high-performance ssds with long lifetimes. In *Proceedings of DAC'15* (San Francisco, CA, Jun 2015).

[13] HARDOCK, S., PETROV, L., AND GOTTSTEIN, R. Database system on ftl-less flash storage. In *Proceedings of VLDB'13* (Riva del Garda, Italy, Aug 2013).

[14] HUANG, P., SUBEDI, P., HE, X., HE, S., AND ZHOU, K. FlexECC: partially relaxing ecc of mlc ssd for better cache performance. In *Proceedings of ATC'14* (San Jose,CA, Feb 2011).

[15] HUANG, P., WU, G., HE, X., AND XIAO, W. An aggressive worn-out flash block management scheme to alleviate ssd performance degradation. In *Proceedings of EuroSys'14* (Amsterdam, Netherlands, April 2014).

[16] JOSEPHSON, W. K., BONGO, L. A., FLYNN, D., AND LI, K. DFS: a file system for virtualized flash storage. In *Proceedings of FAST'10* (San Jose, CA, Feb 2010).

[17] KANG, J.-U., HYUN, J., MAENG, H., AND CHO, S. The multi-streamed solid-state drive. In *Proceedings of HotStorage'14* (Philadelphia, PA, Jun 2014).

[18] KIM, H., AND AHN, S. BPLRU: a buffer management scheme for improving random writes in flash storage. In *Proceedings of FAST'08* (San Jose, CA, Feb 2008).

[19] KIM, Y., ORAL, S., SHIPMAN, G. M., LEE, J., DILLOW, D. A., AND WANG, F. Harmonia: a globally coordinated garbage collector for arrays of solid-state drives. In *Proceedings of MSST'11* (Denvor, CA, May 2011).

[20] LEE, C., SIM, D., HWANG, J.-Y., AND CHO, S. F2FS: a new file system for flash storage. In *Proceedings of FAST'15* (Santa Clara, CA, Feb 2015).

[21] LEE, S., KIM, J., AND ARVIND. Refactored design of i/o architecture for flash storage. In *Computer Architecture Letters* (Jan 2015), vol. 14, pp. 70–74.

[22] LEE, S., LIU, M., JUN, S., XU, S., KIM, J., AND ARVIND. Application-managed flash. In *Proceedings of FAST'16* (Santa Clara, CA, Feb 2016).

[23] LEFURGY, C., RAJAMANI, K., RAWSON, F., FELTER, W., KISTLER, M., AND KELLER, T. W. Energy management for commercial servers. *Computer Archive 36*, 12 (Dec 2003), 39–48.

[24] LI, W., JEAN-BAPTISE, G., RIVEROS, J., NARASIMHAN, G., ZHANG, T., AND ZHAO, M. CacheDedup: in-line deduplication for flash caching. In *Proceedings of FAST'16* (Santa Clara, CA, Feb 2016).

[25] LIM, H., FAN, B., ANDERSEN, D. G., AND KAMINSKY, M. Silt: A memory-efficient, high-performance key-value store. In *Proceedings of SOSP'11* (Cascais, Portugal, Oct 2011).

[26] LIU, R.-S., YANG, C.-L., AND WU, W. Optimizing nand flash-based ssds via retention relaxation. In *Proceedings of FAST'12* (San Jose, CA, Feb 2012).

[27] MANNING, C. Yaffs: yet another flash file system.

[28] MARMOL, L., SUNDARARAMAN, S., TALAGALA, N., , RANGASWAMI, R., DEVENDRAPPA, S., RAMSUNDAR, B., AND GANESAN, S. Nvmkv: A scalable and lightweight flash aware key-value store. In *Proceedings of HotStorage'14* (Philadelphia, PA, Jun 2014).

[29] MIN, C., KIM, K., CHO, H., LEE, S.-W., AND EOM, Y. I. SFS: random write considered harmful in solid state drive. In *Proceedings of FAST'12* (San Jose, CA, Feb 2012).

[30] OH, Y., CHOI, J., LEE, D., , AND NOH, S. H. Caching less for better performance: Balancing cache size and update cost of flash memory cache in hybrid storage systems. In *Proceedings of FAST'12* (San Jose, CA, Feb 2012).

[31] OUYANG, J., LIN, S., JIANG, S., HOU, Z., WANG, Y., AND WANG, Y. SDF: software-defined flash for web-scale internet storage systems. In *Proceedings of ASPLOS'14* (Salt Lake City, UT, Mar 2014).

[32] PARK, H., KIM, J., CHOI, J., LEE, D., AND NOH, S. H. Incremental redundancy to reduce data retention errors in flash-based ssds. In *Proceedings of MSST'15* (Santa Clara,CA, May 2015).

[33] QIN, D., BROWN, A. D., AND GOEL, A. Reliable writeback for client-side flash caches. In *Proceedings of ATC'14* (Philadelphia, PA, Jun 2014).

[34] SANDISK. Flash Management: a detailed overview of flash management techniques. http://docplayer.net/11833543-Wp001-flash-management-a-detailed-overview-of-flash-management-techniques.html.

[35] SHEN, Z., CHEN, F., JIA, Y., AND SHAO, Z. Optimizing flash-based key-value cache systems. In *Proceedings of HotStorage'16* (Denvor, CA, Jun 2016).

[36] SOUNDARARAJAN, G., PRABHAKARAN, V., BALAKRISHNAN, M., AND WOBBER, T. Extending ssd lifetimes with disk-based write caches. In *Proceedings of FAST'10* (San Jose, CA, Feb 2010).

[37] TECHTARGET. Flash memory. http://searchstorage.techtarget.com/definition/flash-memory.

[38] TSENG, H.-W., GRUPP, L. M., AND SWANSON, S. Understanding the impact of power loss on flash memory. In *Proceedings of DAC'11* (San Diego, CA, June 2011).

[39] WU, G., AND HE, X. DeltaFTL: improving ssd lifetime via exploiting content locality. In *Proceedings of EuroSys'12* (Bern,Switzerland, April 2012).

[40] ZHANG, Y., ARULRAJ, L. P., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. De-indirection for flash-based ssds with nameless writes. In *Proceedings of FAST'12* (San Jose, CA, Feb 2012).

[41] ZHENG, M., TUCEK, J., QI, F., AND LILLIBRIDGE, M. Understanding the robustness of ssds under power fault. In *Proceedings of FAST'13* (San Jose,CA, Feb 2013).

[42] ZUCK, A., TOLEDO, S., AND SOTNIKOV, D. Compression and ssd: Where and how? In *Proceedings of INFLOW'14* (Broomfield, CO, Oct 2014).