# A Software-Defined Fusion Storage System for PCM and NAND Flash

Zheng Li*, Shuangwu Zhang*, Jingning Liu†‡, Wei Tong†, Yu Hua†, Dan Feng*†, and Chenye Yu*
* Wuhan National Lab for Optoelectronics, Wuhan 430074, China
† School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
{lizheng, zhang25, jnliu, Tongwei, csyhua, dfeng, yuchenye} @hust.edu.cn

***Abstract*—In SSD-based storage systems, persistent data are stored in the NAND Flash and however manipulated in DRAM, causing the decoupled inefficiency. The data being closer to the processors are much easier to be lost due to the volatile property of DRAM, leading to serious data reliability problems. In the meantime, existing SSD technology exploits the properties of NAND Flash and leverages NAND Flash + controller + FTL architecture to improve system's performance. In this black-box-modeled structure, the data semantic information is hard to be transferred by conventional interface. Hence the SSD firmware fails to make full use of the performance potential of SSD in terms of the semantic information of data. Moreover, the host can't obtain physical characteristics and statistical information about SSD, failing to be used by the file system or I/O scheduling algorithm designed for the disks.**

**In order to address these problems, we propose a software defined fusion storage system for PCM and NAND Flash. PCM can be defined as the same level storage or as a buffer of NAND Flash to reduce the WA (Write Amplification) of Flash and improve reliability. In this system, we expose the channels, erases counts and data distribution of PCM/NAND Flash to the host, design FTL algorithm close to file system to obtain more semantic information of data accessing, and manage the storage device as a non-transparent structure. To achieve these design goals, we implement a Host Fusion Storage Layer (HFSL) that supports flexible I/O schedule algorithm and address mapping of variable allocation size in the persistent superior performance of SSD. Extensive experimental results demonstrate the efficiency of the proposed schemes.**

***Keywords*—Fusion, NAND Flash, PCM**

## I. Introduction

System reliability is quite important but difficult to be predictable. For example, in October 2012, data centers in New York were broken down and resulted in power outages due to the occurrence of the hurricane, which led to serious data loss and caused an unprecedented disaster for the data center industry [1]. In August 2012, the data centers of Shanda [2] (a company in China) suffered from inefficient performance, including data loss and hard recovery, due to disk damages. The reliability problems for data have caused the damage to companies and institutions. In conventional computer systems, DRAM is used as the main memory. With the increase of memory size and the processing power, data availability, integrity and consistency in DRAM become tricky to guarantee. Fortunately, the emerging of new non-volatile memory technologies such as phase change memory (PCM), resistive random access memory (RRAM), and magnetoresistive random access memory (MRAM) provide a possible solution to address these problems. Among these technologies, PCM is the most promising one to replace main memory [3]. Unfortunately, PCM exhibits 10 times slower relative to DRAM and it is more worse in real industry production. However, its storage density as well as bit addressable have the salient advantages owing to the non-volatile property. These characteristics allow PCM to replace DRAM in some applications to meet the needs of large capacity and non-volatile reliability.

Currently, most SSDs are encapsulated into a black box, using the NAND Flash + controller + FTL architecture. However, the use of this architecture in SSD cannot be fully exploited in terms of the NAND Flash's bandwidth. The host cannot obtain the equipment information and statistical information (such as read/write/erase count, data placement) of NAND Flash. The host cannot effectively combine the accessing data's semantic characteristics with the physical characteristics of the SSD to enhance performance [4][5]. In order to address these problems, existing work mainly uses Host-Based SSD, such as Fusion-io, OFTL, FPGA-Based SSD. Specifically, Fusion-io puts forward a VSL (Virtual Storage Layer) [6] in the host driver to manage the SSD, and designs a Direct File System [7] for its SSD to improve the performance. Another FPGA-Based SSD [8] prototype platform leverages a fast and reconfigurable algorithm in software that reorganizes the traditional emulation infrastructure by moving the FTL to the host software instead of keeping it in the FPGA-based flash interface. In addition, the host inside FTL structure is widely used in embedded devices, such as JFFS [9], YAFFS [10], UBIFS [11]. These three file systems are designed according to NAND Flash characteristics. In order to mitigate write amplification from file systems, OFTL [12] is proposed to be co-designed with flash memory to reduce write amplification. In terms of the disk access latency, the total time consumed in software I/O stack of an I/O request can be overlooked, which however can't be ignored in SSD due to the low latency of NAND Flash. In order to address these problems, SDF [13] proposes a method that reduces the software overhead via a well-designed user-space interface and thin driver. The host software can explore the SSD's raw potential for performance improvements.

In order to address the problem of data reliability in SSD and maximize performance, we propose a software-

†‡ Corresponding author: Jingning Liu (jnliu@hust.edu.cn)

978-1-4673-6688-5/15/$31.00 

TABLE I: Comparison of Software-Defined Fusion Storage System with Existing SSD Structures

| | Design Goals | FTL Location | I/O Path | Interface | Experiment | References |
|---|---|---|---|---|---|---|
| Traditional SSD | Flash's High Performance | Device | Long | Block I/O | Prototype | Micron SSDS[14] |
| FPGA-based SSD | Reliability/Endurance | Host | Short | Page Unit R/W | Prototype | FCMM[8] |
| Fusion-io SSD | Application Acceleration | Host | Long/Short | Page Unit R/W, Block I/O | Prototype | Fusion-io report[6] |
| Object-based FTL | FS's write amplification | Host | Short | Bytes/Page Unit R/W | Simulation | FAST[12] |
| Software-defined Flash | Raw flash's bandwidth and usable capacity | Device | Short | Pages Unit R/W | Prototype | ASPLOS[13] |
| Our design | SSD is transparent to Host, SSD's data reliability/security | Host | Short | Bytes/Page Unit R/W | Prototype | |

defined fusion storage system for PCM and NAND Flash and implement a prototype platform to demonstrate the efficiency of the fusion storage system. We compare the software-defined storage system with the state-of-the-art systems as shown in Table I. The software-defined storage system puts FTL in host, like FPGA-based SSD, Fusion-io SSD and OFTL. Unlike them, its FTL is closer to file system and can exploit the accessing data feature easily. In our system, all the physical characteristics of PCM and NAND Flash including the channel numbers, channel state, R/W latency, erasing count and statistics information are exposed to the host. The system can both provide R/W interface in byte and page granularity. In addition, the next-generation memory chips — PCM are adopted in the fusion system. In this design, the PCM can be used not only to be the NAND Flash's buffer, but also the same level storage of NAND Flash to store metadata or small write data. Two key features of the methodology are:

**HFSL**: To achieve the design goals, we design a **H**ost **F**usion **S**torage **L**ayer (HFSL) in the host. In HFSL, we implement flexible I/O scheduling and management strategy via exploiting the physical properties from PCM and NAND Flash, and simplify the overall architecture of the storage device. HFSL translates requests from file systems or applications into simple operations such as read, write and erase, and sends them to the device. Hence, the device only needs to deal with simple requests without address translation, garbage collection, wear-leveling, the redistribution or combination of the I/O request. The application layer only needs to call the interface provided by HFSL to access PCM or NAND Flash directly without the use of I/O subsystem.

**Fusion method**: In software-defined storage system, we fuse NAND Flash with PCM. Physical addresses of PCM and NAND Flash are addressed together. In the system, PCM could act as the same-level storage as NAND Flash to store metadata information and small write data. Meanwhile, PCM can be used as the NAND Flash's buffer, to improve the data reliability and shorten the R/W lantency. PCM and NAND Flash fusion methods become flexible and adaptive according to access patterns. If the accessed data are important and large-size, the data are first translated to the PCM and then written to the NAND Flash. If the accessed data are metadata or small-size data from file systems or applications, the data are written to PCM directly. PCM can reduce NAND Flash's write amplification caused by the storage of metadata and small data.

We implement the software-defined fusion storage system prototype independently and design the HFSL to manage the system. The prototype uses a carrier card and several sub cards physical structure, the carrier card is connected to the host through PCI-e interface. The software-defined storage structure has some salient features:

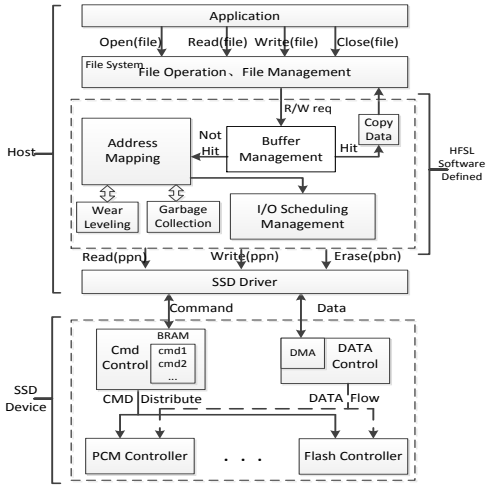



Fig. 1: The system architecture

1) Uses a software-defined method to fuse PCM with NAND Flash. PCM can be defined as the same level storage as NAND Flash or the NAND Flash's buffer.
2) Uses a non-transparent structure. The physical characteristics and statistical information of the storage device are exposed to the host.

The rest of this paper is organized as follows. Section II describes the system design, including the HFSL and hardware designs. Section III presents and analyzes experimental results. Section IV introduces the related work. Finally, Section V concludes our paper.

## II. The System Design

### A. Design Overview

The architecture of software-defined fusion storage system for PCM and NAND Flash, as shown in Figure 1, mainly includes host based HFSL and the fusion PCM/NAND Flash device. HFSL is designed to manage the SSD as a non-transparent structure for software to better use of the physical properties. The existing of HFSL makes it possible for users to define their method of data storing. PCM can be defined as the same level storage as NAND Flash to store small-size data, metadata to enhance the performance of small write or be defined as the NAND Flash's buffer to improve data reliability. We design an allocation granularity variable address mapping and efficient I/O scheduling algorithm in HFSL. These two algorithms directly obtain the file system information, manage the software-defined fusion storage system via the physical structure and improve the performance of the system. In this section, we put forward a solution with an actual prototype. In the prototype system, the device provides two interfaces: the command interface and data interface. The driver offers read, write, erase function interfaces to HFSL and can directly

call command interface to transmit the commands into the device by passing traditional complexity command path. HFSL provides several commands such as open, close, read/write operating in page or bytes unit, and can convert the file system commands into physical operations of NAND Flash or PCM.

### B. HFSL Design

The HFSL mainly includes address mapping and I/O scheduling algorithms and acts as a middle layer between the host and device. In this section, we will introduce the design of HFSL detailedly.

*1) **Address Mapping**:* The properties of NAND Flash and PCM are greatly different. Because of NAND Flash's erase-before-write property, it needs to use an addressing table to perform the virtual-to-physical address translations. Unlike NAND Flash, PCM supports in-place update, but its write endurance is still limited. The PCM also needs to maintain an addressing table to avoid some PCM cells being worn out quickly.

TABLE II: The status flags in address mapping table

| Name | Description |
|---|---|
| Counter | LPN write counter for judgment of hot or cold |
| State | To identify which sub-LPN are stored in PCM |
| Flag | To identify whether the table entry is changed |
| Add_Flag | LPN mapping to in Flash or PCM |

HFSL adopts a page-level address mapping algorithm and the address mapping table of NAND Flash and PCM is kept in the fixed field of PCM to improve the reliability. As shown in Figure 2, PCM and NAND Flash are addressed unified, but the addressing sizes of the PCM and NAND Flash are different. NAND Flash uses the page as its addressing size while PCM uses 512B to provide flexible operation interface. Different addressing granularities bring enormous flexibility for the system. In the Figure 2, LPN is reffered to Logical Page Number and PPN is reffered to Physical Page Number. One LPN in Flash will be devided into four sub-LPN(LPN1,LPN2,LPN3,LPN4) in PCM. Other status flags of the addressing table are showed in Table II.

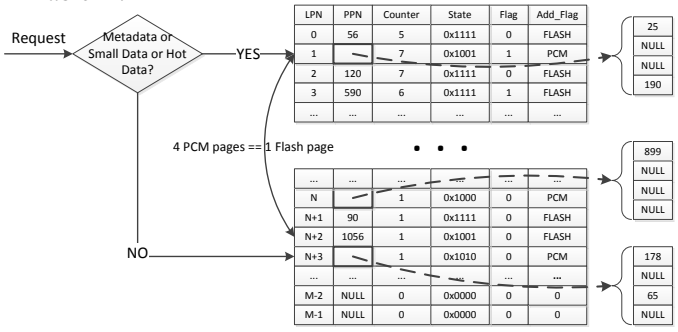



Fig. 2: The address mapping structure

In order to reduce the write amplification of Flash, the metadata and small write data can be saved to PCM. The threshold size of the small write ($SSW$) can be defined by software according to application requirements. And the address mapping algorithm has three strategies: a) if the accessing request size is smaller than $SSW$, the request will be allocated to PCM physical address. If the PCM space is not enough, the cold data on the PCM should be moved to NAND Flash, and then the address mapping table will be modified immediately; b) count the wearing times of PCM and NAND Flash. In the address allocation process, if the data of the request are cold, they will be allocated into physical block with higher wearing times; c) in order to improve the request's parallelism, HFSL uses a stripe addressing method and the continuous logical address would be distributed to different channels. These above strategies improve the small write performance as well as the lifespan of Flash, provide an opportunity for software to be awared of the physical structure and make better decisions to utilize physical parallelism.

TABLE III: The Command Fields

| Abbreviation | Description |
|---|---|
| Reserved | Reserved Space |
| Command ID | Command's number |
| op | The command's operation type |
| flag | The command's complete status |
| ppn | The physical page number |
| buffer | The data address in main memory |

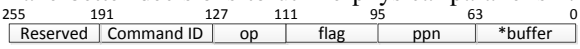



Fig. 3: Command format

*2) **I/O Scheduling Algorithm**:* The I/O scheduling algorithm is simple but effective. HFSL creates a cycling request queue for each channel and distributes the requests that are sent to each channel. Requests would be packaged as the command format showed in Figure 3 and be sent to the device. The explanation of each command field is shown in the Table III. The I/O scheduling algorithm is shown in Figure 4. The incoming request is added to the tail of the circular queue and the head of the circular queue will be moved forward to the next node location when a request is completed. The data counter records the number of requests which is not completed. When the timer comes, the I/O scheduling algorithm will send requests to the device. In order to reduce the I/O times, the algorithm will send $n$ ($n = 8-255$, $n$ is configurable) requests to the device once. If the length of the queue is less than n, all the requests in the queue will be sent to the device. The data counter will increase the number of requests that have been sent. When a completed interrupt happens, the request execution status will be checked. If the request flag is marked as completed, the head will move forward to the next request node and return a completion to the host after all the completed requests are handled. And the data counter decrease by the number of the completed requests.

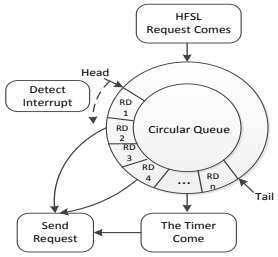
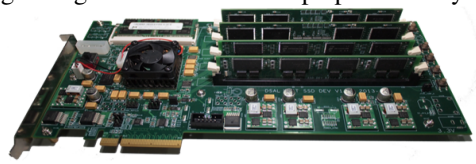



Fig. 4: I/O scheduling structure

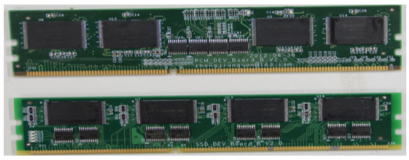
### C. Hardware Prototype

In order to design a flexible fusion structure, we adopt a physical structure of sub card and carrier card, including one carrier card and four sub cards. The carrier card is connected with the host PCIe interface and is made up of Xilinx Virtex-6 FPGA, DRAM, BRAM, PCI-e and other hardware embedded resources. There are four 240 PIN DIMM interfaces in the carrier card, used to connect with the sub cards, as illustrated in

Figure 5(a). The sub card has two types, as illustrated in Figure 5(b), the above one is a PCM card consisting of eight PCM chips provided by Micron. The other is a NAND Flash card consisting of eight NAND Flash chips provided by Samsung.



(a) Carrier Card



(b) Sub Card

Fig. 5: The software-defined fusion storage hardware

In the prototype, we design a high performance data transmission algorithm, including the request processing, the data transmission and NAND Flash/PCM controller. Figure 6 describes the hardware structure of PCM and NAND Flash fusion prototype system. Request processing is used to process the data and control information acquired from the host. The data transmission is used for DMA transmission between the controller and the host's memory. And the controller is used primarily to controll Samsung NAND Flash and Micron PCM chips respectively. Moreover, with the carefully designed controller, data can be transmitted directly between host memory and NAND Flash or PCM, which simplifies the data transmission path within the device.

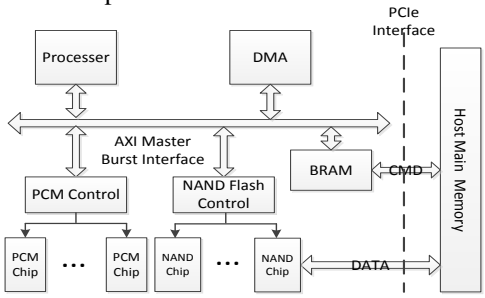



Fig. 6: The hardware structure

## III. Evaluation

In this section, we evaluate the benefits of our prototype named DSAL-SSD, which has been deployed in the computer server. We are going to answer the following questions:

1) How does Flash benefit from the proposed software defined storage techniques? Lifespan or performance?
2) DSAL-SSD places some data in PCM. Is it harmful? What is the bad influence on PCM?

Specifically, we first present the experimental parameters and the construction of the experimental environment, and then analyze the results of separate write modes.

### A. Experimental Setup

In the hardware prototype, the ratio of NAND Flash and PCM can be configured according to user's requirements. The parameters of the experimental environment are described in Table IV. And Using four real workloads with unique characteristics, such as Finacial1[15], MSNFS[16], RADIUS[16], shown in TableV, to evaluate the benefits of software defined storage system.

TABLE IV: Parameters of Fusion Storage Prototype System

| Parameter | Value |
|---|---|
| CPU | Intel Xeon E5-2620@2GHz, 6 Cores |
| OS | Linux 2.6.32 kernel |
| Host Interface | PCIe 1.1X4 |
| Flash / PCM Channel Count | 8/2 |
| Flash Page Size | 2KB |
| Flash Chip Size | 2GB |
| PCM Chip Size | 16MB |

TABLE V: Benchmarks

| Benchmarks | Description |
|---|---|
| Trace1 | Most are small requests, from Finacial1 |
| Trace2 | Many repeated write requests, from Finacial1 |
| Trace3 | Most are heavy requests, from MSNFS |
| Trace4 | A mix of small and heavy requests, from RADIUS |

### B. Flash Update Times

Flash's read and write operations can only be executed in page granularity. What's worse, much larger portions of flash must be erased or rewritten than actually required, and this is called Write Amplification (WA). The Write Amplification will lead to the increase of the writing and wearing times of NAND Flash and the decrease of the performance and lifespan of SSD.

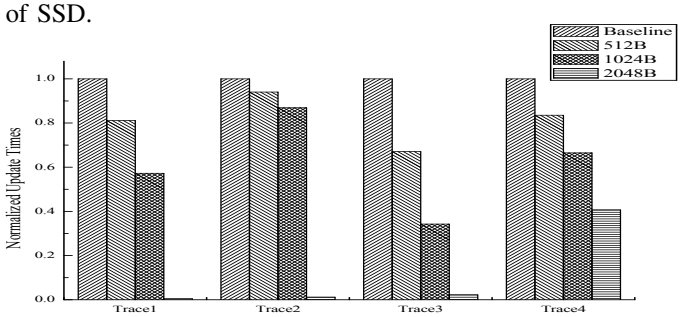



Fig. 7: The Normalized Flash Update Times

To evaluate the benefits of reducing the WA times in the proposed software defined strategies, we measure the update times of Flash by varying the $SSW$ from 512B, 1024B to 2048B. The results are illustrated in Figure 7. The update times of prototype without PCM are set as the baseline. In the figure, we can get two conclusions: First, as $SSW$ increases, the update times of Flash reduce significantly compared with baseline. When $SSW$ is set as the size of a page, 2KB, there is almost no update because small requests are written to PCM. Second, the update wouldn't be disappeared even though $SSW$ is the same as page size as showed in Trace4. That is because some LPNs are already stored in NAND Flash, and if these LPNs are accessed again by small write requests, the requests will first read the old data form NAND Flash. Then the accessing data will be merged with the old data and then written to PCM. In read domain workload, update is the bottleneck of the performance, especially for read performance and the reduction of WA can significantly improve the performance as well as lifespan of NAND Flash.

### C. Flash Write Times

Flash memory can only be programmed and erased within a limited number of times. This is often referred to as the

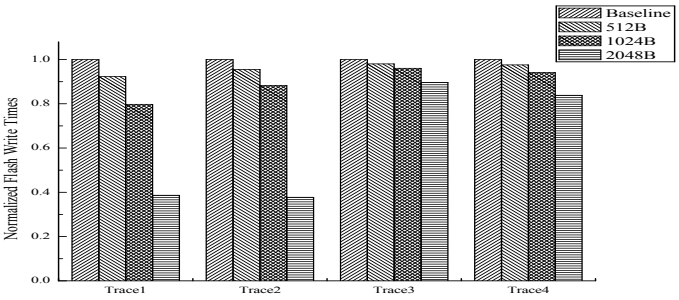



Fig. 8: The Normalized Flash Write Times

maximum number of program/erase cycles (P/E cycles) it can sustain over the life of the flash memory. Frequent program and erase operations will cause serious lifespan problem. As shown in Figure 8, the use of PCM decreases Flash's write times significantly. Specially, in Trace1 and Trace2, the write times decrease to almost 60%. Moreover, the small write performance of PCM is much higher than Flash, and the write latency is much shorter. The reducing of Flash write times can be beneficial to the write performance as well as the P/E times.

### D. Flash Read Times

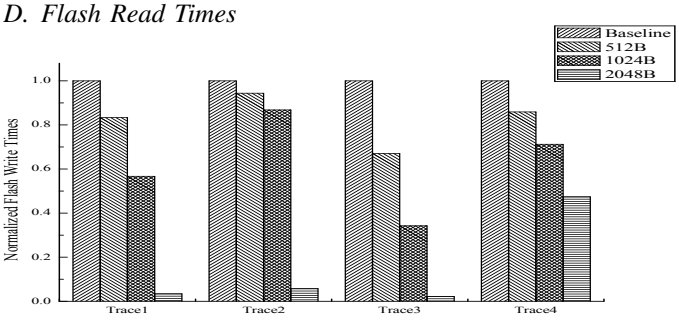



Fig. 9: The Normalized Flash Read Times

Figure 9 shows the Flash Read Times with different $SSW$. In the figure, the Flash read times reduce from 10%-90%. Especially in Trace3, the read times reduces sharply, because there is no read requests in Trace3 and all read operations are resulted from WA. When $SSW$ is set to 2KB, there is almost no WA, so the read times reduce sharply. Otherwise, the size of write requests in Trace1 and Trace2 is largely 1KB-2KB big, so if $SSW$ is larger than 1KB, the effect will be remarkable. The small read performance of PCM is much higher than NAND Flash. What's more, the critical read delay becomes shorter, hense the whole performance will be improved.

### E. PCM Write Times and PCM Read Times

The using of PCM reduces small writes of the NAND Flash and the WA times while extending the lifespan and improving the performance of NAND Flash, but all these benefits are based on writing data into PCM. Although the storage system benefits from the advantages of bit addressable, in-place update, high read and write performance under small reads and writes, the damages to PCM can't be ignored.

In this section, we measure the PCM read and write times in different $SSW$ and the result is illustrated in Figure10 and Figure11. As show in Figure11, read times of PCM increases with $SSW$ grows. Like NAND Flash, the read operation of PCM is non-destructive, so it won't deliver any disadvantages on lifespan. Figure10 shows the write times of PCM as $SSW$

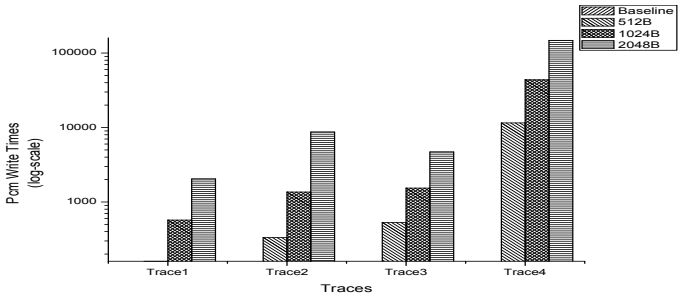



Fig. 10: PCM Write Times for Different Benchmarks

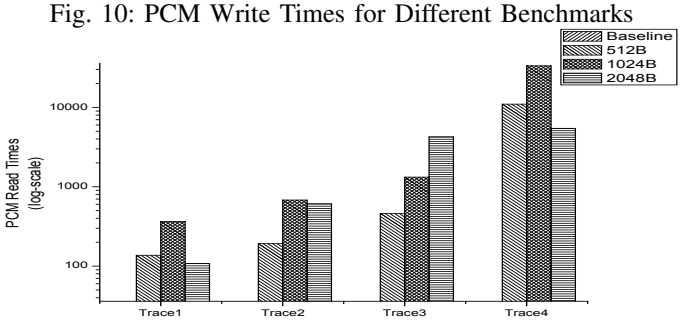



Fig. 11: PCM Read Times for Different Benchmarks

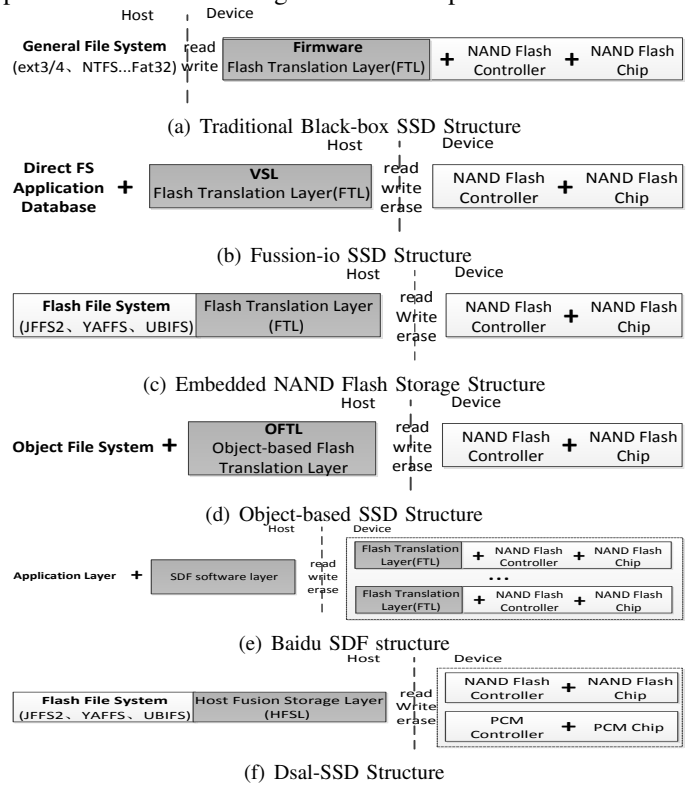
varies. However, considering the huge endurance gap between PCM and Flash($10^9 \gg 10^5$), the bad influence on lifespan of PCM is insignificant. It is worthy to improve the whole storage system performance and latency introducing PCM to complete small read and write requests.

## IV. RELATED WORK

### A. The SSD Solution

Fusion-io puts forward a host-side SSD storage structure, as illustrated in Figure 12(b). Compared with traditional SSD [14] as illustrated in Figure 12(a), Fusion-io SSD designs FTL in the host driver program forming a new host side FTL layer: Virtual Storage Layer (VSL). The VSL provides an address space, and users can deploy a file system on this address space and run the application program or database software directly. The Fusion-io SSD handles the task which originally needs to be completed in the SSD firmware in the host driver program. However, the Fusion-io SSD still encapsulates the SSD to be a black box, and the user can hardly improve the system performance. Host-based FTL structure is widely used in embedded devices, as illustrated in Figure 12(c), and in server accelerator card scheme [17], [18]. In addition, an object-based flash translation layer (OFTL) [12] is proposed to extend the service life and improve efficiency of the SSD. The OFTL merges metadata information to reduce the write amplification from the file system, shown as Figure 12(d). Baidu proposes a Software-Defined Flash (SDF) [13] to allow the host accessing flash directly, shown as Figure 12(e). The SDF takes full advantage of the flash's advantages by its hardware/software co-designed storage system. SDF exposes individual flash channels to the host software and makes a lot of performance optimization according to their application environment and SSD's physical information. Figure 12(f) shows our storage structure. Our design implements a Host Fusion Storage Layer (HFSL) that supports flexible I/O schedule algorithm and allocation size variable address mapping. Via HFSL, the SSD exposes the channels, erases count and data distribution of PCM/NAND Flash to the host, designs

FTL algorithm close to file system, and manages the SSD as a non-transparent structure. The host software, accessing to the PCM/Flash and defining the method data stores, can effectively organize its data while taking full use of the SSD's potential performance and reducing the Write Amplification of Flash.




(a) Traditional Black-box SSD Structure

(b) Fussion-io SSD Structure

(c) Embedded NAND Flash Storage Structure

(d) Object-based SSD Structure

(e) Baidu SDF structure

(f) Dsal-SSD Structure

Fig. 12: Structures of Flash-based storage system

### *B. The Applications of PCM*

Most research work focuses on the study of PCM as a nonvolatile memory, storage class memory, SSD's buffer or utilized as a data protection measure. UCSD laboratory researchers develop a high performance storage array system — Moneta prototype system [19], which makes full use of the excellent performance of PCM, and puts forward a high performance storage structure. Lee, Hyung Gyu [20] puts forward an Energy- and Performance-Aware DRAM/PCM hybrid architecture, researching various of DRAM/PCM hybrid configurations, to significantly improve the system performance without affecting the power. And G. Dhiman [21] proposes a hybrid memory system by combining DRAM with PCM. In addition, PCM can be combined with NAND Flash, using the PCM's excellent merits to improve storage performance.

## V. Conclusion

In SSD-based storage system, the essential metadata information and address mapping table are easy to be lost, hence it is difficult to guarantee the reliability of the information. Moreover, the black-box SSD is transparent to host, and thus host can't make full use of the NAND Flash's performance potential. In this paper, we design a software-defined fusion storage system for PCM and NAND Flash, and implement a Host Fusion Storage Layer (HFSL) using for managing the system. In the HFSL, we implement address mapping of variable allocation size and flexible I/O scheduling algorithm, to support the persistent superior performance of the system. Moreover, PCM can not only be used to store the metadata or address mapping table information, but also be used as the NAND Flash's buffer. By leveraging PCM, the system obtains benefits of lifespan extension, performance improvement. We demonstrate the effective of fusion storage system by the real benchmarks running on our actual hardware prototype.

## Acknowledgment

This work was supported by the National Basic Research 973 Program of China under Grant No.2011CB302301; 863 Project No.2015AA015301, No. 2015AA016701, No.2013AA013203; NSFC No.61303046, No.61472153, No.61173043; The Fundamental Research Funds for the Central Universities, HUST:2013TS042. This work was also supported by Key Laboratory of Information Storage System, Ministry of Education, China.

## References

[1] "Hurricane sandy takes data centers offline with flooding, power outages," http://arstechnica.com/information-technology/2012/10/.

[2] "Shanda games," http://www.shandagames.com/us-en/index.html.

[3] B. C. Lee *et al.*, "Architecting phase change memory as a scalable dram alternative," in *Proc.ISCA*, 2009, pp. 2–13.

[4] Y. Hua, X. Liu, W. He, and D. Feng, "Design and implementation of holistic scheduling and efficient storage for flexray," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 25, no. 10, pp. 2529–2539, October 2014.

[5] D. Zhan, "Spatiotemporal Capacity Management for the Last Level Caches of Chip Multiprocessors," in *in Ph.D. Dissertation Thesis, Computer Engineering, University of Nebraska-Lincoln*, Dec. 2012.

[6] "Leveraging host based flash franslation layer for application acceleration," http://flashmemorysummit.com/.

[7] W. K. Josephson *et al.*, "DFS: A File System for Virtualized Flash Storage," *TOS*, vol. 6, pp. 85–100, 2010.

[8] Y. Cai *et al.*, "FPGA-Based Solid-State Drive Prototyping Platform," in *FCCM*, 2011, pp. 101–104.

[9] D. Woodhouse, "JFFS: The journalling flash file system," in *Ottawa Linux Symposium*, vol. 2001, 2001.

[10] "Yaffs: Yet another flash file system," http://www.yaffs.net/.

[11] "Ubifs: Unsorted block image file system," http://www.linux-mtd.infradead.org/doc/ubifs.html/.

[12] Y. Lu *et al.*, "Extending the lifetime of flash-based storage through reducing write amplification from file systems," in *FAST*, 2013, pp. 257–270.

[13] J. Ouyang *et al.*, "SDF: software-defined flash for web-scale internet storage systems," in *ASPLOS*, 2014, pp. 471–484.

[14] "Micron ssds," http://www.micron.com/solutions/client-ssd-storage.

[15] "Umass trace repository," http://traces.cs.umass.edu/index.php.

[16] "Snia, block i/o traces," http://iotta.snia.org/tracetypes/3.

[17] Y. Chang *et al.*, "An adaptive file-system-oriented FTL mechanism for flash-memory storage systems," *TECS*, pp. 1–19, 2012.

[18] J. Kim *et al.*, "FlashLight: A Lightweight Flash File System for Embedded Systems," *TECS*, pp. 1–23, 2012.

[19] A. M. Caulfield *et al.*, "Moneta: A high-performance storage array architecture for next-generation, non-volatile memories," in *MICRO*, 2010, pp. 385–395.

[20] H. G. Lee *et al.*, "An energy-and performance-aware DRAM cache architecture for hybrid DRAM/PCM main memory systems," in *ICCD*, 2011, pp. 381–387.

[21] G. Dhiman *et al.*, "PDRAM: a hybrid PRAM and DRAM main memory system," in *DAC*, 2009, pp. 664–669.