Journal of Systems Architecture 000 (2016) 1–18

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

A user-visible solid-state storage system with software-defined fusion methods for PCM and NAND flash

Zheng Li, Fang Wang, Jingning Liu*, Dan Feng, Yu Hua, Wei Tong*, Shuangwu Zhang

Wuhan National Lab for Optoelectronics, Huazhong University of Science and Technology, Wuhan 430074, China

A R T I C L E I N F O

Article history:
Received 3 November 2015
Revised 28 August 2016
Accepted 31 August 2016
Available online xxx

Keywords:
NAND flash
PCM
User-visible structure
Fusion storage system

A B S T R A C T

Existing SSD technology exploits the properties of NAND flash and leverages NAND flash with a controller running FTL algorithms to improve system performance. On one hand, however, in this black-box-modeled structure, data semantic information is hard to be transferred and interpreted by conventional interfaces. Hence, SSD firmware fails to make full use of the performance potential of SSD by utilizing semantic information. Moreover, the host cannot 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. On the other hand, 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. What's more, restrictive read/program granularity and out-of-place updates limit the performance while flash suffers from small size operations.

In order to address these problems, we propose a user-visible solid-state storage system with software-defined fusion methods for PCM and NAND flash. PCM is used for improving data reliability and reducing the write amplification of NAND flash as PCM shows some outstanding features, such as in-place updates, byte-addressable, non-volatile properties and better endurance. In this system, we manage the storage device as user-visible structure rather than black-box-modeled structure. In detail, we expose the number of channels, erase counts and data distribution of PCM/NAND flash to the host and design FTL algorithm closer to file system to obtain more semantic information of data accessing. PCM can be software-defined as the same level storage or buffer of NAND flash to reduce the WA (Write Amplification) of NAND flash and improve the data reliability. Moreover, some key software components (such as FTL, I/O scheduling and buffer management) are also reconfigurable and operated easily combined with physical characteristics. To achieve these design goals, we implement a Host Fusion Storage Layer (HFSL) and redesign the lengthy I/O path. Applications or filesystem can access PCM/flash directly via provided interfaces by HFSL without passing traditional I/O subsystem. Moreover, we provide the system management software to make the storage system can be easily software-defined by the upper-level system. We implement our software-defined fusion storage system in our actual hardware prototype and extensive experimental results demonstrate the efficiency of the proposed schemes.

1. Introduction

System reliability is quite important but difficult to be predictable. For example, in October 2012, data centers in New York were broken down due to power outages, 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] suffered from inefficient performance, including data loss and hard recovery, due to disk damages. The reliability problems for data have caused the damage to both 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 and integrity in DRAM are hard to guarantee. System suffers from unpredictable metadata loss due to the power failure and volatile property of DRAM cache [3]. 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 possible solutions to address these problems. Among these technologies, PCM is one of the most promising candidates for DRAM-based main

* Corresponding authors.
E-mail addresses: lizheng@hust.edu.cn (Z. Li), jnliu@hust.edu.cn (J. Liu), Tongwei@hust.edu.cn (W. Tong).

http://dx.doi.org/10.1016/j.sysarc.2016.08.005
1383-7621/© 2016 Elsevier B.V. All rights reserved.

Table 1
Comparison of software-defined fusion storage system with existing SSD structures.

	Design goals	FTL	I/O path	Interface	Experiment
Traditional SSD [18]	Compatible and high performance SSD	Device-based	Long	Block I/O	Prototype
FPGA-based SSD [12]	Reliability and endurance improvement	Host-based	Short	Page unit R/W	Prototype
Fusion-io SSD [10]	Application acceleration	Host-based	Long/Short	Page unit R/W, Block I/O	Prototype
Object-based FTL [11]	Reducing write amplification of FS	Host-based	Short	Bytes/Page Unit R/W	Simulation
Software-defined Flash [17]	Raw flash's bandwidth and usable capacity	Device-based	Short	Pages unit R/W	Prototype
Our design	User-visible structure and data reliability of SSD	Host-based	Short	Bytes/Page Unit R/W	Prototype

memory [4]. Unfortunately, PCM exhibits 10 times slower compared with DRAM and it is worse in actual industrial production (such as Micron P5Q and P8P PCM chips) [5–7]. However, its storage density as well as non-volatile property has salient advantages compared with DRAM. 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 NAND flash with a controller running FTL algorithms architecture. However, this rigid architecture cannot fully exploit of high bandwidth and random access performance of NAND flash. The host cannot obtain the equipment and statistical information of NAND flash (such as read/write/erase count, data placement), which makes it difficult for host to effectively combine semantic characteristics of accessed data with physical characteristics of SSDs to maximize system performance [8,9]. In order to address these problems, existing work mainly uses Host-Based SSD, such as Fusion-io [10], OFTL [11], FPGA-Based SSD [12]. Specifically, Fusion-io [10] puts forward a middle layer named VSL (Virtual Storage Layer) in the host driver to manage the SSD and designs a Direct File System [13] for its SSD to improve the performance. FPGA-Based SSD [12] leverages a fast and reconfigurable algorithm in software that reorganizes the traditional emulation infrastructure by moving FTL algorithms to host software instead of keeping it in device-side FPGA. In addition, host-side FTL structure is widely used in embedded devices and designed in filesystems such as JFFS [14], YAFFS [15], UBIFS [16]. These three file systems are designed according to NAND flash characteristics and try to provide fast I/O performance. Moreover, In order to mitigate write amplification from file systems, OFTL [11] is proposed to be co-designed with flash memory to reduce write amplification. In terms of disk access latency, the total time consumed in software I/O stack of an I/O request can be overlooked, which, however, cannot be ignored in SSD due to the low latency of NAND flash. In order to address these problems, SDF [17] tries to reduce the software overhead via well-designed user-space interface and thin driver. By exploring the software and hardware co-designed architecture, applications can make full use of the raw bandwidth potential for performance improvements.

In order to address the data reliability problem in SSD and maximize system performance, we propose a user-visible fusion storage system for PCM and NAND flash and implement a hardware prototype to demonstrate the efficiency of the proposed designs. We compare our software-defined fusion storage system with the state-of-the-art systems, as shown in Table 1. Our software-defined storage system is hardware and software co-designed and we modify the entire software layers to make best use of our proposed storage system. In detail, it uses host-side FTL design similar to FPGA-based SSD, Fusion-io SSD and OFTL. Unlike them, we manage the system with user-visible structure and design its FTL closer to the file system to exploit the accessed data features, which can significantly maximize the system performance and reduce the write amplification of NAND flash. In addition, next-generation memory chips, i.e. Phase Change Memory (PCM) chips, are adopted in the fusion system for outstanding features such as in-place updates, byte-addressable, non-volatile properties and better endurance compared with NAND flash. In our system, all the physical characteristics of PCM and NAND flash, such as the number of channels, each channel state, NAND flash or PCM R/W latency, erase count and statistics information are exposed to the host software. PCM can be software-defined as the same level storage or buffer of NAND flash to reduce the WA (Write Amplification) of NAND flash and improve data reliability. Moreover, some key software components (such as FTL, I/O scheduling and buffer management algorithms) are also reconfigurable and operated easily combined with physical characteristics. Moreover, the lengthy I/O path is redesigned to reduce the overall software overhead for conventional I/O path is not suitable for the properties of NAND flash and PCM. Our fusion storage systems can both provide byte/page granularity R/W interfaces. Applications or filesystem can access NAND flash and PCM directly via provided interfaces without passing conventional I/O subsystem. Moreover, we provide the system management software to make the storage system can be easily software-defined by the upper-level system. For example, applications can configure the storage system by setting the parameters provided by system management software.

Two key features of our design are:

HFSL: To achieve the proposed design goals, we implement a novel **H**ost **F**usion **S**torage **L**ayer (HFSL) in the host-side. In HFSL, we implement flexible I/O scheduling, PCM and NAND flash management strategies via exploiting the physical properties from PCM and NAND flash (such as the number of channels and each channel state). In addition, we significantly simplify the overall architecture of the storage device and system. HFSL translates requests from file systems or applications into simple hardware operations such as read, write and erase, then sends them to the device directly. Hence, the device only needs to deal with simple commands without dealing with address translation, garbage collection, wear leveling, requests redistribution or combination of the I/O request. By exploring HFSL, we can significantly shorten the I/O path. Applications only need to call the byte or page interface provided by HFSL to access PCM or NAND flash directly without passing the traditional I/O subsystem.

Fusion methods: In our software-defined fusion storage system, we fuse NAND flash and PCM with different software-defined methods. In the system, PCM could act as the same-level storage as NAND flash to store metadata, random or small write data. Meanwhile, PCM can be used as the buffer of NAND flash to improve data reliability and decrease R/W latency. PCM and NAND flash fusion methods are flexible and adaptive to access various 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 will be written into PCM directly. PCM can significantly reduce the write amplification of NAND flash caused by the storage of metadata, random or small write data and improve the system performance as well as the lifespan of NAND flash.

Table 2
Characteristics comparison of memory and storage technologies.

Storage/ Memory	Feature size(F^2)	R/W/E latency	Endurance	Write energy	Idle power	Volatility
DRAM	6–10	$< 10ns(R) < 10ns(W)$	$> 10^{15}$	0.1 nJ/b	$100mW/GB$	YES
PCM	4–8	$55ns(R)$ $150ns(W)$	$> 10^{8}$	0.6 nJ/b	$1mW/GB$	NO
NAND flash	11–16	$25us(R)$ $500us(W)$ $2ms(E)$	$> 10^{15}$	1 nJ/b	$1-10mW/GB$	NO
HDD	–	$5ms(R)$ $5ms(W)$	$> 10^{16}$	2 nJ/b	$10W/TB$	NO

In general, our software-defined fusion storage system has some salient characteristics compared with the existing SSD structures in the following aspects:

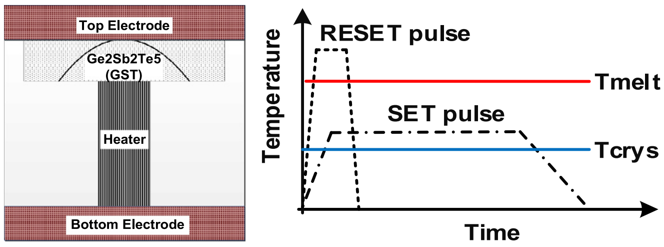
- **User-visible structure**. Benefits from the software and hardware co-designed, the physical characteristics and statistical information of the storage device are exposed to the host in our fusion storage system, unlike conventional black-box-modeled SSDs. The host can effectively combine the semantic characteristics of data with the physical characteristics of SSD to enhance performance. What's more, the fusion methods of PCM and NAND flash are visible to the software. PCM can be defined as the same level storage as NAND flash to store write data, metadata and hot data or the buffer to improve the reliability and small write performance of SSD.
- **Customized software algorithms**. In our software-defined fusion storage system, HFSL is designed in the host-side and independent modules are used for implementing different software components (such as FTL, I/O scheduling and Buffer management). We can achieve special software algorithms to meet the various requirements of applications such as performance, reliability without changing hardware architecture. Moreover, the overhead of I/O data path is extremely high when using high-performance SSDs [19]. In our design, file systems or applications can access PCM and NAND flash directly by exploring flexible interfaces provided by HFSL without passing traditional I/O subsystem. The length of I/O path is shortened and overall software overhead is reduced also.
- **Actual hardware prototype**. We implement our software-defined fusion storage system in our actual hardware prototype independently and we evaluate the effectiveness and efficiency of proposed design in the actual hardware platform. The prototype adopts a carrier card and several sub cards physical structure, where the carrier card is connected to the host via a PCI-e interface. The flexible carrier-sub card structure makes the software-defined fusion storage system more flexible and configurable in the capacity ratio and the number of channels of PCM and NAND flash.

The rest of this paper is organized as follows. Section 2 describes the system design, including HFSL and hardware designs. Section 3 presents and analyzes experimental results. Section 4 introduces the related work. Finally, Section 5 concludes this paper.

2. The system design

2.1. Background

DRAM-based main memory has reached its scalability bottleneck and it is hard to achieve high capacity and low energy consumption under 22nm [20]. Fortunately, non-volatile Memory (NVM) such as Phase Change Memory (PCM), Magnetoresistive RAM (MRAM), Ferroelectric RAM (FeRAM) and Resistive Random Access Memory (RRAM) have better scalability with lower power consumption. Among these non-volatile memory technologies, PCM is the most important technology under industrial application, which allows PCM to be one of the most promising candidates for DRAM-based main memory [4,21,22].



(a) PCM cell structure (b) write process

Fig. 1. PCM cell structure with two independent electrodes connecting GST and the heater. Quickly heat under high temperature (above the melting point) and quench the glass can make the PCM cell amorphous. Otherwise, holding the glass above its crystallization point but below the melting point for a while makes the PCM cell into the crystalline state.

PCM exploits the unique behavior of resistance of chalcogenide glass, typically the alloy of germanium, antimony and tellurium (GST, commonly Ge2Sb2Te5) [23]. The resistance of GST varies hugely between crystalline and amorphous states. The amorphous glass has high electrical resistance while the crystalline state shows extremely low resistance. Thus, current value varies much under the same voltage level between amorphous and crystalline state, which can be used to present the digital information.

Typical PCM cell structure is shown in Fig. 1(a) [4,21–23]. In general, a PCM cell consists of a top electrode and a bottom electrode. Between these two electrodes are the heater and chalcogenide glass (GST). Through the heating element, we can make the PCM cell amorphous via quickly heat under high temperature (above the melting point) and quench the glass. Instead, holding the glass above its crystallization point but below the melting point for a while can make the PCM cell into the crystalline state. Fig. 1(b) shows the detailed write mechanisms of PCM [4,21–23].

Table 2 shows the characteristics of PCM compared with other memory and storage technologies (DRAM, NAND flash and Hard Disk Drive (HDD)). Noticing that the parameters vary much from different prototypes, the parameters in this table are from the surveys of various literatures [4,20–22,24,25]. We compare these memory and storage technologies in various aspects, such as feature size, R/W latency, endurance, write energy, idle power consumption and volatility.

Compared with NAND flash, PCM shows great potential in scalability for the excellent feature size ($4-8F^2$ in PCM compared with $11-16F^2$ in NAND flash). What's more, write performance of PCM is much higher than NAND flash (50 ns in PCM compared with 25 us in NAND flash). More importantly, PCM supports in-place update that updates can happen in place and it does not have to allocate a new copy of the data compared with NAND flash. In short, PCM has many advantages over NAND flash and HDD while its performance is close to DRAM, which suffers from high refresh energy consumption and volatile property. PCM can be used to store persistent data and has great potential to be one of the most promising candidates of DRAM-based main memory.

2.2. The concept of software-defined

Software-defined storage (SDS) provides a new approach for storage management. The key idea of SDS is to decouple the control software and the hardware it is managing. In other words, the underlying hardware is abstracted out like storage virtualization and the software enables policies for managing the storage and data services. With SDS, storage devices are virtualized and managed as a storage pool. Administrators can be freed from the heterogeneous and complex data storage management, just working with the management interfaces provided by SDS.

Baidu SDF (Software-defined Flash) aims to take full advantage of the performance advantages of SSD by its hardware and software co-designed storage system [17]. SDF exposes individual flash channels to the host software and makes a lot of performance optimization according to their web applications and physical information of SSDs. The host software can make full use of the parallelism of multi-channels and explore the raw performance potential for service time reduction.

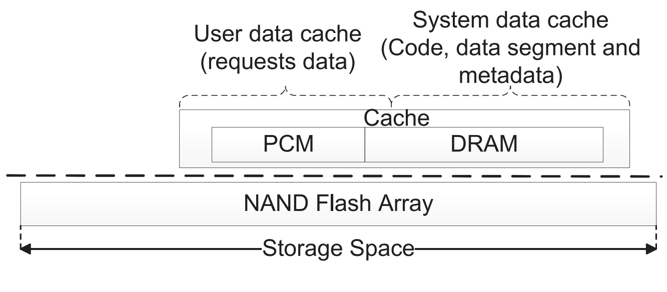
SDS and SDF aim at the separation of hardware devices and software control layers. Similar but different, our software-defined fusion focuses on the following aspects:

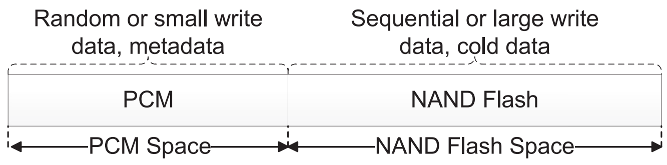
- **Hardware and software separation**. Our software-defined fusion system utilizes the key idea of hardware and software separation. Compared with SDF, our design exposes not only the channels number but also the physical characteristics and statistical information of the storage device. The host can effectively combine the semantic information of accessed data with the physical characteristics of SSD to enhance performance and reduce the write amplification of NAND flash.
- **Special software algorithms**. The I/O scheduling algorithms and address mapping granularities are software-defined according to demands of applications. We can change algorithms without changing the hardware device. By adjusting mapping granularity dynamically, we can maximize the performance of various applications for PCM provides byte granularity interface and better random or small write/read performance.
- **Software-defined storage method**. The fusion methods of PCM and NAND flash are software-defined. PCM can be defined as the same level storage as NAND flash to store small write data, metadata and hot data or the buffer of NAND flash to improve the overall system performance and data reliability of SSD.
- **Customizable FTL**. In our software-defined fusion storage system, FTL is designed in host-side and we use independent modules to implement different FTL components (address mapping, cache management, wear leveling and garbage collection). We can achieve special FTL algorithms to meet the requirements of performance, security, lifespan etc..

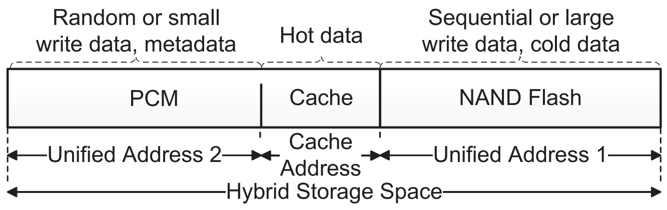
2.3. Fusion methods

There are three fusion methods in our software-defined fusion storage system. PCM can be configured as the upper level cache, the same level storage or both the upper level cache and the same level storage. In this section, we will introduce these fusion methods respectively.

- **PCM as the upper level cache of NAND flash**. PCM can be configured as the cache for NAND flash to cooperate with the traditional DRAM cache for its properties of non-volatile, low power consumption etc.. As shown in Fig. 2(a), PCM and DRAM compose the cache of NAND flash. Part of the PCM/DRAM space is used for the code segment, data segment and metadata of NAND flash while the remaining space is used for caching write data. By exploiting the non-volatile property of PCM, we can improve the data reliability and avoid the problem of lost data due to sudden power failure.
- **PCM as the same level storage of NAND flash**. PCM can also be configured as the same level storage of NAND flash. As shown in Fig. 2(b), NAND flash and PCM share the unified addressing space. Software can determine the data distribution according to their data types. On one hand, metadata, random or small write data can be written into PCM for the in-place updates and byte-addressable properties of PCM. On the other hand, it also reduces the write amplification of NAND flash, which leads to the increasing number of write and erase operations, performance degradation and lifespan decreasing of NAND flash.
- **PCM as both the upper level cache and the same level storage of NAND flash**. PCM can be configured as both the upper level cache and the same level storage of NAND flash. As shown in Fig. 2(c), part of PCM space is configured as the cache of NAND flash and the rest space acts as the same level storage. The ratio of cache and storage space is also software-defined by users. On one hand, if there is massive metadata, random or small write data, software can increase the storage space of PCM to store these data and reduce the write amplification of NAND flash. On the other hand, when there is massive hot write data in the requests, we can improve the cache space to reduce the response time.


User data cache (requests data)
System data cache (Code, data segment and metadata)
Cache
PCM
DRAM
NAND Flash Array
Storage Space
(a) PCM as the upper level cache


Random or small write data, metadata
Sequential or large write data, cold data
PCM
NAND Flash
PCM Space
NAND Flash Space
(b) PCM as the same level storage



(c) PCM as both the upper level cache and same level storage

Fig. 2. Software-defined fusion storage methods.

In short, the fusion methods of PCM and NAND flash are flexible and efficient. Software can define the fusion methods according to reliability or performance requirements. For example, software can define the threshold size of the small write (SSW) according to application requirements to maximize system performance and reduce write amplification of NAND flash. The metadata, random or small write data whose size is smaller than SSW can be written

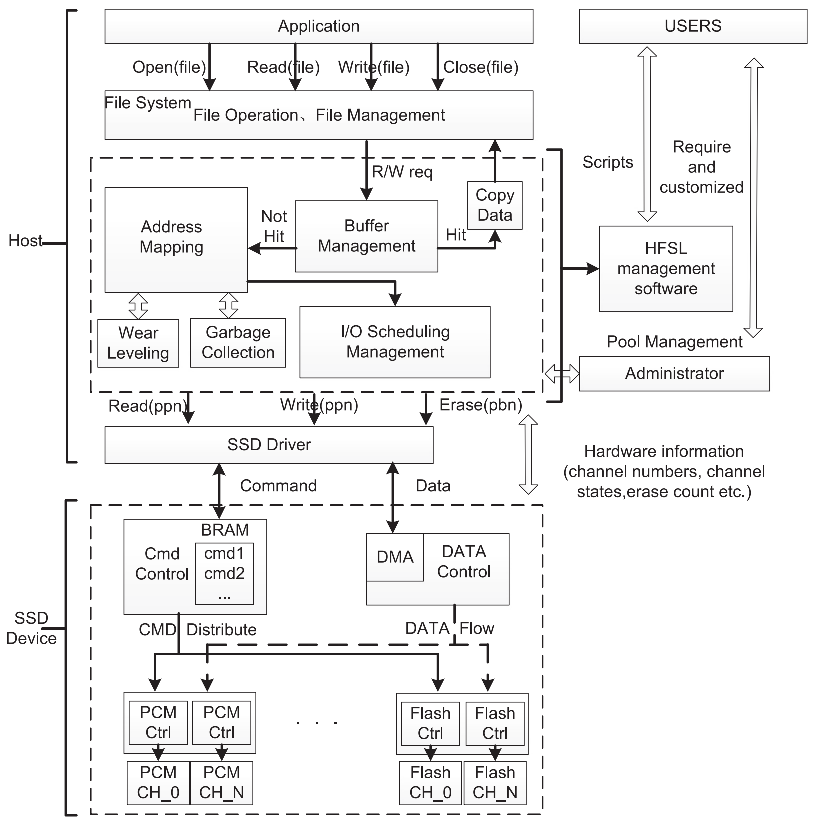


Fig. 3. The system architecture.

to PCM by configuring PCM as the buffer or the same level storage of NAND flash.

2.4. Design overview

The architecture of software-defined fusion storage system for PCM and NAND flash is shown in Fig. 3. The system mainly includes the host-side HFSL and fusion PCM/NAND flash storage devices. HFSL is designed to manage the SSD as a user-visible structure device and provides carefully implemented interfaces for applications to better make use of the physical properties of NAND flash and PCM. HFSL makes it possible for users to define their methods of data storage. In our system, PCM can be defined as the same level storage of NAND flash to store random or small write data, metadata to enhance the performance or be defined as the buffer of NAND flash to improve data reliability. Effective address mapping and efficient I/O scheduling algorithms are implemented in HFSL. These algorithms can be easily modified according to the accessing properties of applications. For example, allocation granularities of NAND flash and PCM are variable and software-defined in the address mapping algorithm. These algorithms directly obtain the information of file system, manage the software-defined fusion storage system via the direct interfaces provided by hardware and improve the overall performance and data reliability.

We implement an actual hardware prototype with industrial PCM and NAND flash chips provided by Micron [6] and Samsung [26], respectively. In the hardware prototype, two interfaces are provided to the software layer, i.e. the command interface and data interface. Thin device driver offers read, write and erase function interfaces to HFSL. Applications can directly call a command interface to transmit the commands into the device by passing traditional complexity command path. HFSL provides several commands such as open, close, read and write operating in page or byte unit and converts the file system commands into direct physical operations of NAND flash or PCM. Applications may call the byte or page interfaces provided by HFSL to access PCM or NAND flash directly without passing the traditional I/O subsystem.

2.5. Hardware prototype

By adopting a sub-carrier-card physical structure, we obtain extremely flexible hardware design. The hardware prototype includes one carrier card and four sub cards. The carrier card, which is made up of Xilinx Virtex-6 FPGA, DRAM, BRAM, PCI-e and other hardware embedded resources, is connected with the host via PCIe interface. There are four 240 PIN DIMM interfaces in a carrier card for connecting the sub cards, as illustrated in Figs. 4 and 5(a). The sub card has two types, i.e. PCM sub card and NAND flash card. As illustrated in Fig. 5(b) and (c), the above one is a PCM card consist of eight PCM chips provided by Micron. Another one is a NAND flash card consist of eight NAND flash chips provided by Samsung. The flexible carrier-sub-card physical structure provides strong support for dynamic capacity configuration, we can make customizable capacity as well as the number of channels of PCM and NAND flash by simply changing the number of PCM and NAND flash cards.

In our hardware prototype, a high performance data transmission algorithm is implemented. The components of the hardware prototype are request processing, data transmission, NAND flash and PCM controllers. Fig. 6 describes the SOC structure of the fusion prototype system. Request processing is used to process the data and control information acquired from the host. The data transmission uses DMA between the device-side controller and the host-side memory. Independent controllers are used to control Samsung NAND flash and Micron PCM chips, respectively. Moreover, with the carefully designed controllers, data can be transmit-

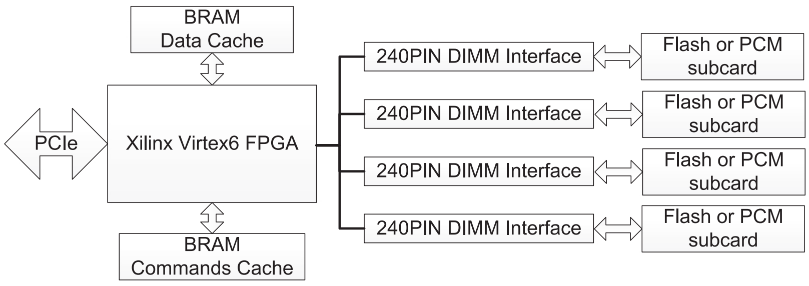


Fig. 4. The hardware structure.

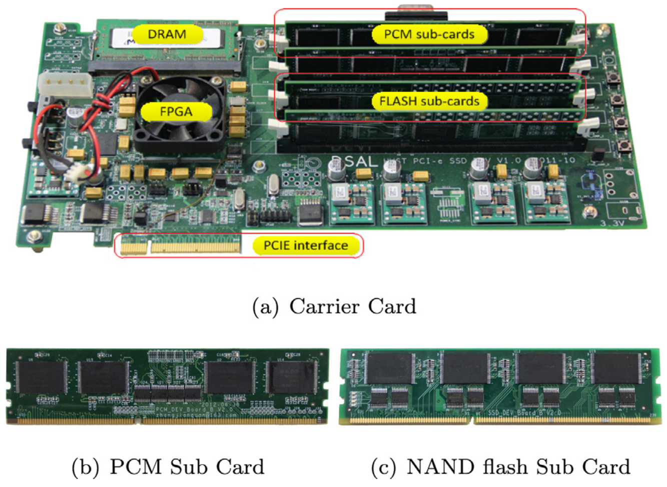


Fig. 5. The software-defined fusion storage hardware.

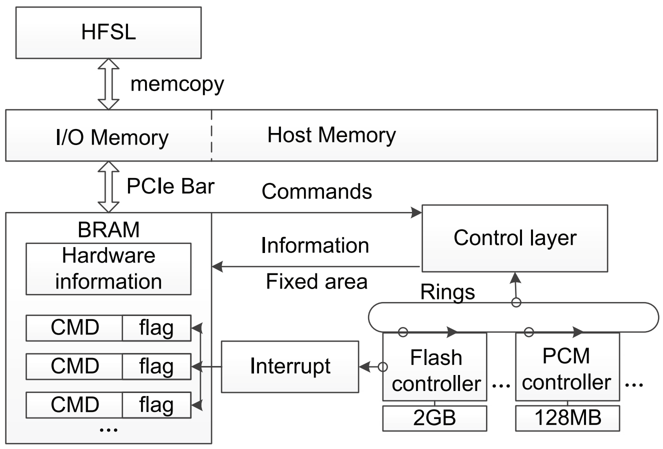


Fig. 7. Firmware.

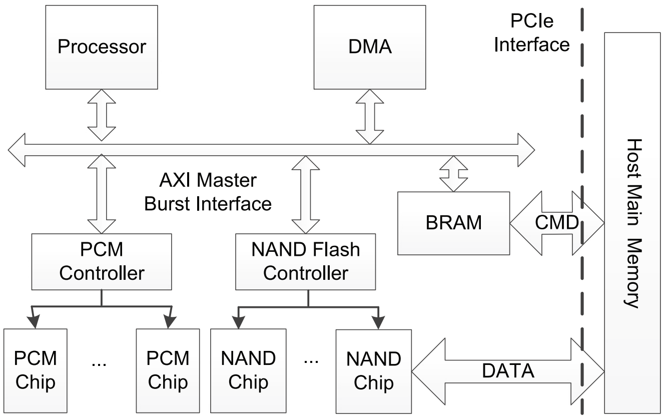


Fig. 6. The SOC structure.

ted directly between host memory and NAND flash or PCM without superfluous data copies, which simplifies the data transmission path with thin device firmware.

The firmware architecture is shown in Fig. 7. As the FTL is moved to the host-side, the key functions of firmware are commands distribution, interrupt control and channels control. Moreover, firmware provides interfaces for host software, i.e. HFSL, to get the hardware information, such as the number of channels, P/E counts for flash etc.. In detail, the BRAM of hardware is mapped in the host memory space and HFSL can access BRAM easily by directly accessing mapped memory area. The hardware information, such as the number of PCM and flash channels, is sent to the fixed area of BRAM via soft core processor. HFSL can get the hardware information easily by accessing this fixed area. Moreover, a command queue is set in the BRAM. Firmware gets commands and delivers them to PCM and flash controllers via sample token rings. Once finished, the flag will be labeled by the interrupt control layer.

2.6. HFSL design

HFSL includes many components such as address mapping, I/O scheduling and buffer management. HFSL acts as a middle layer between the host layer and device layer, which shortens the length of I/O path and reduces the overall software overhead. In this section, we will introduce the design details of HFSL.

2.6.1. Address mapping

The physical properties of NAND flash and PCM are quite different. Due to the erase-before-write property of NAND flash, addressing table is used to perform the virtual-to-physical address translation. Unlike NAND flash, PCM supports in-place update. However, PCM still suffers from limited endurance and it is necessary to avoid some PCM cells being worn out quickly. As we have to maintain an addressing table for NAND flash, we can also design wear leveling method for PCM by utilizing this addressing table with low space and computational overhead.

HFSL adopts a page-level address mapping algorithm and the address mapping table of NAND flash and PCM is stored in the fixed field of PCM to improve the reliability [27]. As shown in Fig. 8, PCM and NAND flash are addressed unified, but the addressing sizes of the PCM and NAND flash are different. NAND flash uses the page size (2 KB in our design) addressing granularity while PCM uses 512 B to provide flexible operation interfaces in the default configuration. Different addressing granularities bring enormous flexibility to the system. As shown in Fig. 8, LPN is referred to Logical Page Number and PPN is referred to Physical Page Number. One LPN in NAND flash will be divided into four sub-LPNs

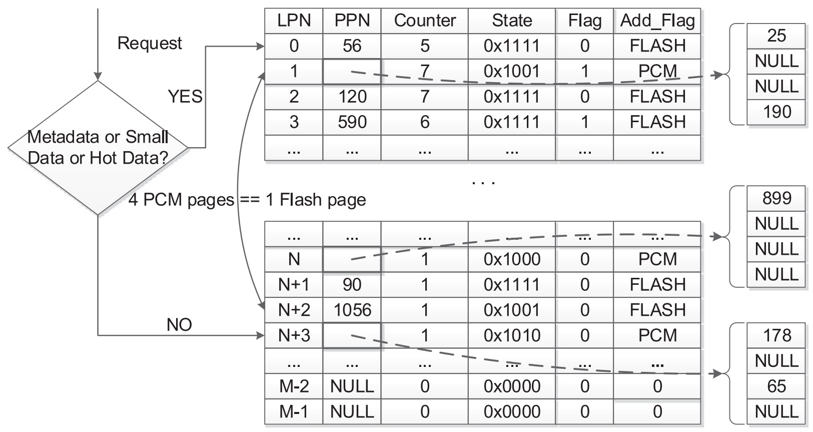


Fig. 8. The address mapping structure.

Table 3
The status flags in address mapping table.

Name	Description
Counter	LPN write counter for hot/cold data judgment
State	To identify which sub-LPN are stored in PCM
Flag	To identify whether the table entry is changed
Add_Flag	To identify whether LPN is mapped in NAND flash or PCM

(LPN1, LPN2, LPN3 and LPN4) in PCM. For example, as shown in Fig. 8, LPN is stored in PCM with Add_Flag indexing it. This LPN will be addressed to four sub-LPNs. PPN 25 and 190 will be contacted with sub-LPN1 and sub-LPN4. Other status flags of the addressing table are showed in Table 3.

In order to reduce the write amplification of NAND flash, metadata, random or small write data are stored in PCM for the flexible operation granularities and better performance. The threshold size of small write (*SSW*) can be defined by software according to the accessing patterns of applications. In HFSL, address mapping strategies are concluded as follows: (a) If the accessing request size is smaller than *SSW* or the flag is labeled with “metadata”, the request will be allocated to PCM physical address. If the PCM space is not enough, cold data on the PCM would be moved to NAND flash and then the address mapping table will be modified immediately; (b) The wear times of PCM and NAND flash will be counted. In the address allocation process, if the requested data are cold, they will be allocated into physical block with higher wearing times; (c) In order to improve parallelism, HFSL uses a stripe addressing method and continuous logical addresses would be distributed to different channels.

Since HFSL using host-side FTL, the algorithm of code data judgment is customizable. In this design, we use the simple unidimensional bloom filter algorithm to present whether data are hot or cold. These above strategies improve the random or small write performance as well as the lifespan of NAND flash, provide an opportunity for software to aware of the physical structure and make better decision to utilize multi-channel parallelism.

Moreover, since hardware is abstracted and visible to the host-side HFSL, the management of metadata, random or small write data in PCM can be implemented easily. As shown in the Fig. 3, the host-side software can access PCM and NAND flash directly by giving physical address. The critical metadata, such as the address mapping table for example, is stored in the fixed area of PCM. When fusion storage system boots, HFSL reads the metadata out of the fixed physical address of PCM using interfaces provided by the driver. When halts, HFSL writes down the metadata to PCM, similarly.

2.6.2. I/O path

Traditional multilevel I/O subsystem includes Block Device Driver, I/O Scheduling, Generic Block Layer and File System, as shown in Fig. 9. The storage media, such as disks and SSDs, stores the data files generated by filesystem, which is implemented to control how data are stored and managed. User applications can call the interfaces provided by the I/O system to access these files. In general, requests are packaged in bio-format and sent to the request queue. After that, the I/O scheduler manipulates all requests, such as bio requests merging, to minimize the latency and maximize the throughput. I/O scheduler sends requests to the device driver after scheduling. Driver converts the bio-format request to the device-side register-based commands and applies a memory region for DMA transmission. Once device finishes the commands, an interrupt is sent to the device driver. Then driver releases the occupied memory region and awakes waiting process.

When we use hard disks, the software overhead is relatively low and can be ignored (0.3% in UCSD technical report [19]). However, when implementing high speed SSDs, the lengthy I/O path can cause latency increment and performance degradation.

In detail, traditional bio request size is 512 B, i.e. the size of a sector. However, read and write operations in NAND flash can only be executed in page granularity (2 KB or bigger), which leads to seriously write amplification problem, i.e. one operation must be finished with many extra read/write operations, which will lead to the increasing of write and erase times of NAND flash as well as performance and lifespan decreasing. To address these problems, traditional SSDs try to merge many bio requests and write them into one NAND flash page to reduce the write amplification. However, requests merging is not the fundamental solution and we need to redesign the request format to combine with the special physical characteristics of NAND flash and PCM. Moreover, traditional I/O scheduling algorithm, such as elevator algorithm, considers the physical properties of disks and tries to get the shortest disk seek-time to improve the overall access performance. However, the physical properties of NAND flash are significantly different from disks. NAND flash has no seek time and random access performance is much better than disks. Traditional I/O scheduling algorithm has no benefit on performance improvement of NAND flash and introduces additional software overhead. It also improves the overall power consumption of the storage system [28].

To address these problems, we redesign the I/O path and try to reduce the software overhead. Our software-defined fusion storage system provides direct read and write interfaces to applications without passing the conventional lengthy I/O path. Our fusion storage system tries to combine I/O scheduling algorithm with physical properties of NAND flash and PCM. Traditional request for-

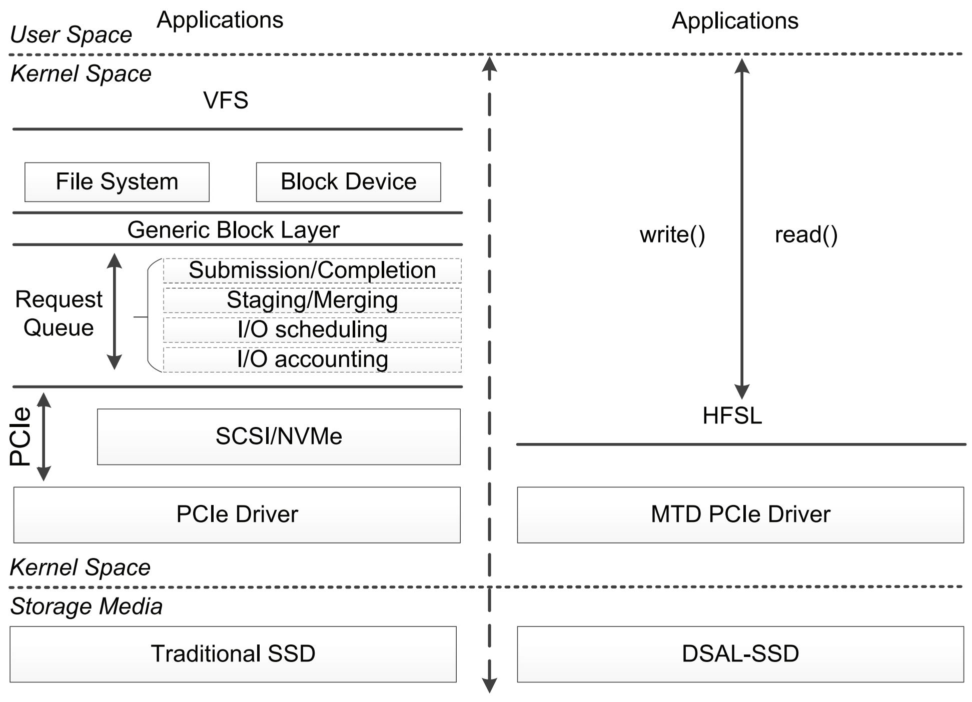


Fig. 9. I/O path.

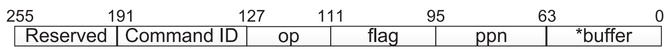


Fig. 10. Command format.

Table 4
The command fields.

Abbreviation	Description
Reserved	Reserved space
Command ID	Command number
op	Command operation type
flag	Command completion status
ppn	Physical page number
buffer	Data address in host-side main memory

mat is too rigid and not suitable for flexible system design. We redesign request format and I/O scheduling algorithm and make the physical information visible to software (the number of channels and data distribution in PCM and NAND flash are exposed to the host software). In detail, HFSL tries to bypass the scheduler in the block layer. We shield the original requests merging and reordering functions in the block layer. We only do some check of received bio requests and try to deal with these requests in HFSL as soon as possible. After that, HFSL calls the driver directly and waits for completion of the requests. Moreover, the whole device can be treated as a raw device. Applications only need to call the read and write interfaces provided by HFSL without utilizing traditional I/O subsystem. As shown in Fig. 9, the I/O path is highly simplified compared with the conventional lengthy I/O path. Our design can significantly reduce the software overhead and take full advantage of high raw performance of NAND flash and PCM.

2.6.3. 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 to the idle channel. Requests are packaged into command format as shown in Fig. 10 and sent to the device directly. The description of command format is shown in the Table 4 and the I/O scheduling algorithm is shown in Fig. 11. The incoming request is added to the tail of the circular queue and the head of the queue will be moved forward to the next node when a request is completed. The data counter records the number of requests that are not completed. When the timer comes, the I/O scheduling algorithm will check the value of counter. Only when the counter value is zero, i.e. all requests sent to device before are finished does scheduler send requests to the device. The data transmission between device-side BRAM and host-side memory is relatively expensive, especially when there are a lot of transmissions with small size data. In order to reduce the I/O transmission times, our algorithm will send n requests (n is configurable, from 8 to 255, and can be configured easily by setting a parameter **MAX_CMD_SEND_COUNTER**, as shown in Table 5) to the BRAM on device each time, as shown in Fig. 7. If the length of the queue is less than n, all the requests in the circular queue will be sent to the device. The data counter will record the number of requests that have been sent. If the number of recorded requests is beyond the capabilities of device, i.e. the circular queue is full, the I/O sched-

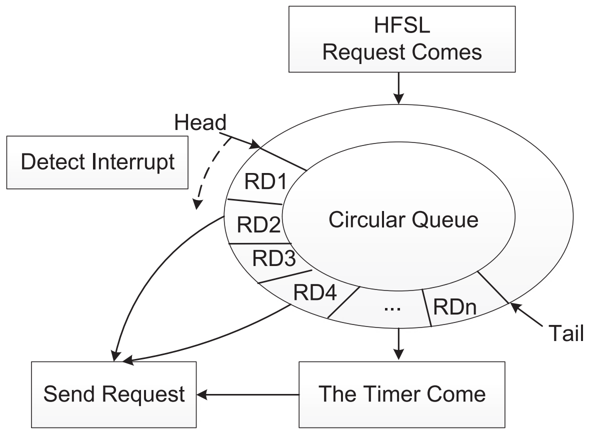


Fig. 11. I/O scheduling structure.

Table 5
Parameters for software-defined storage.

System parameters	Description
FLASHSIZE	Total size of NAND flash
PAGESIZE	Page size of NAND flash
BLOCKSIZE	Block size of NAND flash
BLOCKNUM	The number of blocks in NAND flash
PCMSIZE	Total size of PCM
SUBPAGE_SIZE	Page size of PCM
MAX_CMD_SEND_COUNTER	Maximum length of cycle command queue
Configuration parameters	**Description**
FUSION_METHOD	Software-defined fusion methods
ADDR_GRAN_PCM	Addressing granularity of PCM
ADDR_GRAN_flash	Addressing granularity of NAND flash
SSW	Threshold of randomly write judgment

uler may stop sending requests. When a completed interrupt happens, the execution status of requests will be checked. If the flag is marked as completed, the head will move forward to the next request node. A completion interrupt will be sent to the host after all requests are handled while the data counter will be emptied and ready to deal with coming requests.

2.6.4. Buffer management

Our fusion storage system allows users to define PCM as the buffer of NAND flash. The buffer management algorithm in HFSL creates a buffer region for SSD and manages it efficiently. The buffer management makes full use of the precious memory resources to allocate a big buffer region for caching as much as possible data. The HFSL buffer management utilizes a balanced binary tree to save the buffered nodes and manage them by LRU algorithm, which delivers effective buffer nodes. If there is a request for accessing the SSD, the buffer management will traverse the balanced binary tree according to the logical address and determine whether the data exist in the buffer. If hits, the request will be marked as completed and the data in the buffer will be sent to users. Otherwise, the logical address will be transformed to physical address and the I/O scheduling algorithm will add the request to the tail of the circular queue.

2.7. HFSL management software

The key idea of our fusion storage system is to decouple the integrated storage devices into two individual parts: underlying hardware and management software. By leveraging the management software, the storage device is visible and we can provide customized storage service according to the features of applications.

Our fusion storage system utilizes the host-side FTL designs and provides a lot of services for users or administrators, such as access interfaces for individual NAND flash channels, address mapping, I/O scheduling and fusion methods for PCM and NAND flash. As shown in Fig. 3, HFSL management software is implemented for users and administrators to build or adjust the storage system by software. By leveraging scripts and commands interfaces provided by management software, our fusion storage system can realize three types of software-defined storage methods. As shown in Table 5, the configuration parameters of our fusion storage system are software-defined. The size of NAND flash, PCM as well as other hardware parameters, such as page size, maximum length of command queue, can be set by software.

2.7.1. Software-defined fusion methods

There are three fusion methods for NAND flash and PCM in our software-defined system. To make the fusion methods software-defined, the three fusion methods are encapsulated into different modules. Users or administrators can choose the fusion methods easily by setting the parameter named FUSION_METHOD as shown in Table 5. Different values of FUSION_METHOD present different fusion methods. The management software activates different dealing modules according to the decided results of users or administrators.

Before dealing with a special application, fusion methods should be software-defined first. For example, database or key-value applications have many write requests with small size. Under this circumstance, the fusion methods of NAND flash and PCM shall be defined as the same level storage. PCM can store the data with small granularity and the write amplification of NAND flash is hence reduced. For write dominant workloads, the PCM can be defined as the buffer of the NAND flash and hence reduces improve the endurance of NAND flash.

2.7.2. Software-defined address mapping granularity

In our address mapping algorithm, the key parameter is the threshold size of small write (SSW). The values of SSW determine where the requested data shall be placed and served. Similar to the software-defined fusion methods, the values of SSW are also software-defined by modifying the parameter ADDR_GRAN. The minimal write granularity of Micro P8P PCM chip in our hardware prototype is 2 B size. Moreover, Micro P8P PCM chips provide a buffer write interface, the write granularity is to 64 B. In our hardware design, 8 PCM chips compose a PCM channel, the optimal write granularity is 512 B per channel. In HFSL software-defined layer, ADDR_GRAN can be set from 2 B to 512 B. Based on the features of different applications, users or administrators can set the values of SSW without changing the hardware designs. For some database applications, SSW can be set to 128 B or lower to reduce the write amplification of NAND flash. For some web-scale applications, such as search engine, the throughput is very important. However, the write performance of Micro P8P PCM at 2 KB granularity is lower than NAND flash according to our experimental results. SSW can be set to 512 B to meet the performance demands.

2.7.3. Customizable FTL

In HFSL, FTL algorithms are implemented at the host-side. Address mapping, buffer management, wear leveling etc., are released in different modules. There are two ways to redefine the FTL algorithms.

- The FTL modules can be organized (added or removed) according to the requirements. For some file additional written applications, it is not necessary to do the garbage collection operations. Users can choose to deactivate the garbage collection module, which can improve the overall effectiveness and efficiency of the storage system.
- HFSL is host-based in nature and the hardware is visible to the software. In our fusion storage system, hardware provides direct operation interfaces to the host. For example, the APIs for controlling NAND flash and PCM channels, the number of P/E operations and the state of channels. Administrators can take advantage of these information to release special FTL algorithms combined with the file system. In detail, these FTL algorithms are packaged into individual module. Administrators shall change the module code first and rebuild the module to change the FTL algorithm.

2.7.4. Write protection

Applications can access the storage system directly by using provided read and write interfaces and it causes potentially dangerous problem. In other words, data owned by different applications are easily disturbed intentionally or unintentionally. Proposed software defined storage system can be configured with or

Table 6
Parameters of fusion storage prototype system.

Parameter	Value
CPU	Intel Xeon E5-2620@2 GHz, 6 Cores
OS	Centos 6.5, Linux 2.6.32 kernel
Host interface	PCIe Gen1 X4
FPGA	Xilinx Virtex6 xc6vlx240tff1156-1
NAND flash / PCM channel count	8/2
Flash / PCM chips per channel	2/8
Flash type	SLC
Flash / PCM interface	Toggle DDR / P8P
NAND flash page size	2 KB
Flash block size	128 KB
NAND flash chip size	2 GB
PCM chip size	16 MB

without file system and we proposed two simple policies to avoid "stray writes". On one hand, the write protection can be released by leveraging rights managements with the help of filesystem. User cannot access files stored on other users' directories. On the other hand, only those applications that deployed by the administrators can access the physical addresses directly. These applications are specific and faithful.

3. Evaluation

In this section, we evaluate the benefits of our prototype named DSAL-SSD[1], which has been deployed in the computer server. We are going to answer the following questions:

- What does NAND flash benefits from the proposed software-defined fusion storage techniques? Lifespan or more?
- DSAL-SSD places some metadata, random or small write data in PCM. What is the influence on PCM?
- HFSL shortens the lengthy I/O path, how much performance improvement can get?

Specifically, we first present the experimental parameters and construction of the experimental environment. After that, we discuss the performance improvement delivered by software optimization. Then we analyze the results with different software-defined SSW size under various traces.

3.1. Experimental setup

In the hardware prototype, the capacity ratio and fusion method of NAND flash and PCM can be configured by users according to the data accessing patterns. In this study, PCM is software-defined as the same level storage of NAND flash and the values of SSW are also software-defined. The parameters of the experimental environment are described in Table 6. We use four well-known workloads with unique characteristics, such as Finacial1 [29], MSNFS [30], RADIUS [30], as shown in Table 8. We use these real-world traces collected from different real-world applications and build a simple user-mode test program to evaluate the benefits of the proposed software-defined fusion storage system. Moreover, we use purely page-mapped FTL algorithm in our design. The key ideas behind our design are quite simple. On one hand, page-mapped FTL algorithm shows much better access performance as well as good garbage collection behavior than block-based or hybrid FTL algorithms. On the other hand, HFSL uses host-side FTL architecture, memory sources are abundant to store the mapping table.

[1] DSAL is the abbreviation of our lab, **D**ata **S**torage and **A**pplication **L**ab.

Table 7
Latency of PCM and NAND flash in different granularities.

Granularity	PCM		NAND flash	
	READ(us)	WRITE(us)	READ(us)	WRITE(us)
128 B	12.8	20.7	–	–
256 B	24.6	45.9	–	–
512 B	45.9	93.5	–	–
1024 B	–	–	–	–
2048 B	–	–	75.5	213.2

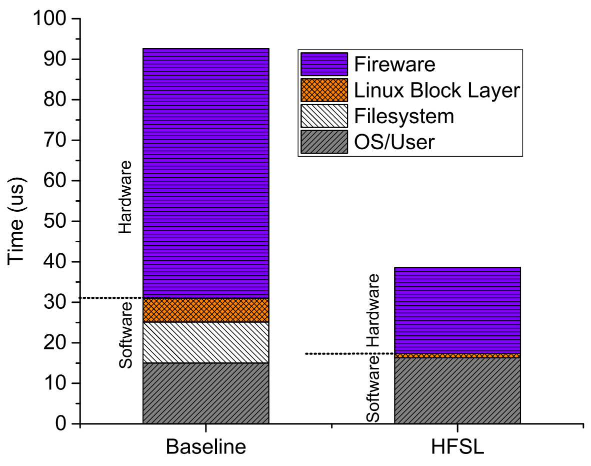


Fig. 12. Latency reduction from optimizations.

3.2. Latency

3.2.1. Latency of PCM and NAND flash

In our design, individual PCM and NAND flash controllers manipulate operations of actual PCM and NAND flash chips. Moreover, data can be transmitted directly between the host memory and NAND flash or PCM controllers without superfluous data copies, which simplifies the data transmission path with thin device firmware. We measure latency of PCM and NAND flash chips in our design with different write granularities. NAND flash can only be operated at page granularity. When it needs to perform operations whose size is less than the page size, we need to fill some invalid data to the command interface of flash chips. Hence the delay remains the time of a page operation. Differently, PCM supports fine-gained operation granularities. In fact, the Micro P8P PCM chips support 2 B up to 512 B write granularities [6,7]. As shown in Table 7, the average delay of NAND flash is 75.5 us and 213.2 us for read and write, respectively. As comparisons, PCM shows much better performance with fine-gained granularities. PCM controller shows 12.8 us, 24.6 us and 45.9 us read latency with 128 B, 256 B and 512 B granularities. The write latency is longer than read, shows 20.7 us, 45.9 us and 93.5 us with 128 B, 256 B and 512 B granularities, respectively. Moreover, we also measure the erase time of NAND flash. Based on our experimental result, erase operation takes more than 1400us on average. Write and erase operations consume large operation time and are the bottleneck to improve the performance of NAND flash.

3.2.2. Latency reduction from optimizations

Fig. 12 shows latency reduction of HFSL design compared with the baseline without any optimization. We introduce the overall time consumptions of the software layer in the host and the firmware in the hardware-side FPGA. In general, the software overhead in host is reduce by 45% and processing time in soft core processor is reduced by 80%. On the one hand, HFSL moves the FTL to host-side and device only needs to deal with simple requests

Table 8
Benchmarks.

Benchmarks	Description	WR	RR	WS (KB)	RS (KB)
Finacial1	Most are random or small size write accesses, from Finacial1 [29]	0.59	0.41	2.91	2.48
Finacial2	Many repeated write requests, from Finacial1 [29]	0.84	0.16	2.49	0.27
MSNFS	Most are sequential or large size write accesses, from MSNFS [30]	0.15	0.85	10.06	14.38
RADIUS	A mix of small and large size write access, from RADIUS [30]	0.91	0.09	7.20	6.58

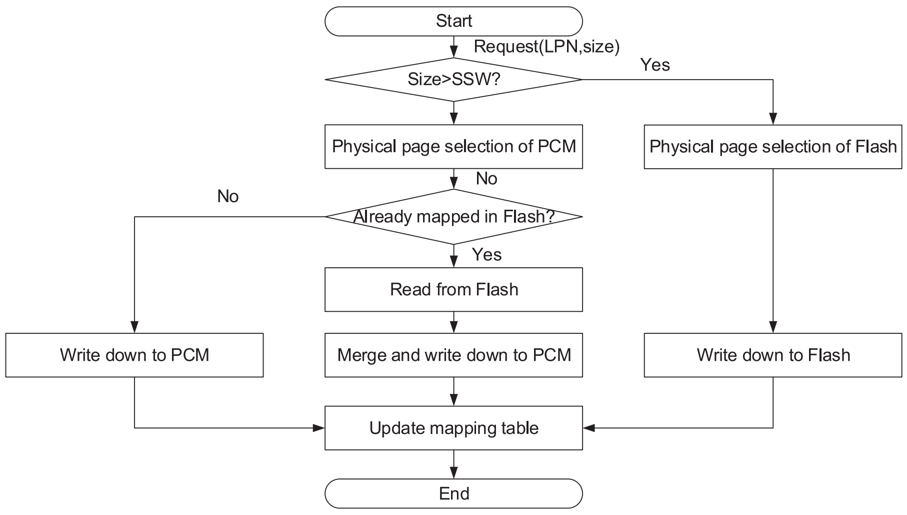


Fig. 13. Software-defined scheme.

without dealing with some complex work such as address translation, garbage collection, wear leveling, requests redistribution and combination of the I/O request. Especially for the low-performance MicroBlaze soft core processor we adopted in our hardware prototype. On the other hand, HFSL significantly shortens the I/O path. The device can be treated as a raw device and applications only need to call the byte or page interfaces provided by HFSL to access PCM or NAND flash directly without passing the traditional I/O subsystem.

3.3. Read and write performance evaluation

3.3.1. Software-defined schemes

We first introduce the software-defined scheme we adopted for evaluation. To evaluate the read and write performance by leveraging PCM storing randomly written data, we use software-defined SSW values (512 B, 1024 B and 2048 B). Moreover, we try to measure the number of updates, write and erase operations of NAND flash as they are the obstacle of performance and endurance improvement. The software-defined scheme is shown in Fig. 13. Since the read and write performance of PCM is much better than NAND Flash at small granularities (when the size of request is less than 2048 B), as shown in Table 7, we try to leverage this asymmetry to boost the system performance. In general, if the size of the request is smaller than SSW, the request is targeted as small write and written down to PCM. Otherwise, data shall be written down to NAND flash. It is noticing that if LPN exists in the mapping table and already be mapped to a PPN in NAND flash, we need to read the data from this PPN and merge the data with coming request.

3.3.2. Trace analysis

We first introduce the experimental results of the write size (WS) distribution of all traces in Table 8. As shown in Fig. 14, the experimental results show that the write size of requests is relatively small in Financial1 and Financial2 (smaller than 4 sectors size on average). Quite differently, there exists numerous heavy requests from MSNFS and a mix of small and heavy requests in RADIUS. In addition, there are many repeated write requests in Financial2. Moreover, all traces have different write or read ratio (WR or RR). Financial2 and RADIUS are write-dominant while MSNFS is read-dominant.

3.3.3. Flash update times

Read and write operations of NAND flash can only be executed in page granularity. What's worse, much larger portions of flash must be erased or rewritten than actually required, which is called Write Amplification (WA). Write amplification will lead to many extra operations and increase of the writing and wearing times of NAND flash, which causes significant performance and endurance degradation.

To evaluate the benefits of decreasing the WA times in the proposed software-defined strategies, we measure the extra operation times of NAND flash named update times by varying the *SSW* from 512 B, 1024 B to 2048 B. The results are illustrated in Fig. 15. The update times without PCM are set as the baseline. We can get two conclusions from the results: First, with *SSW* increasing, the update times of NAND flash reduce significantly compared with the baseline. When *SSW* is set to the size of a page, i.e. 2 KB in our system, there is almost no update because most of small requests are written to PCM. Second, the update would not disappear even though *SSW* is the same as page size as showed in RADIUS. This is because some LPNs are already stored in NAND flash and if these LPNs are accessed again by small write requests, we have to read the old data from NAND flash first. Then the new data will be merged with old data and rewritten to PCM. In read-dominant workload, these extra operations are the bottleneck of performance, especially for sensitive read requests. Moreover, the reduction of WA can significantly improve the performance as well as the lifespan of NAND flash.

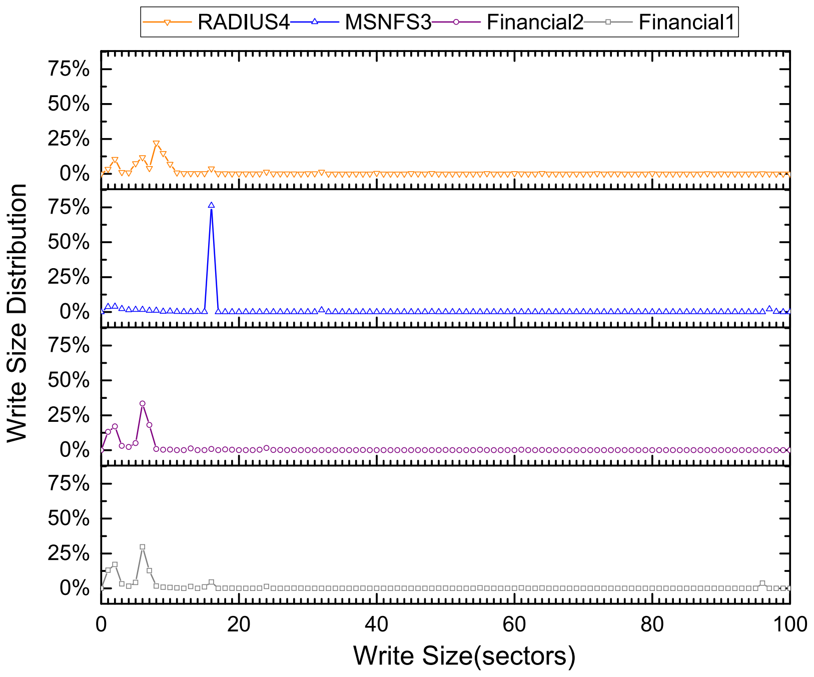


Fig. 14. The normalized write size distribution.

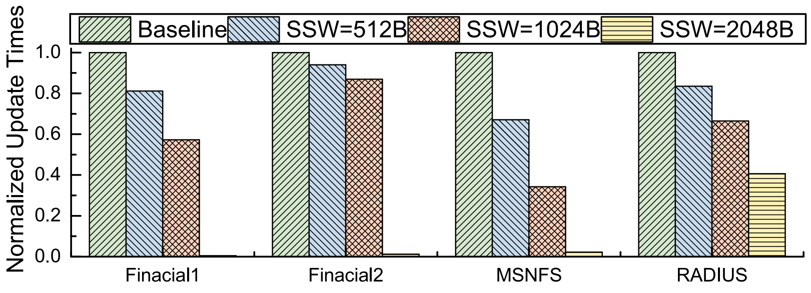


Fig. 15. The normalized flash update times.

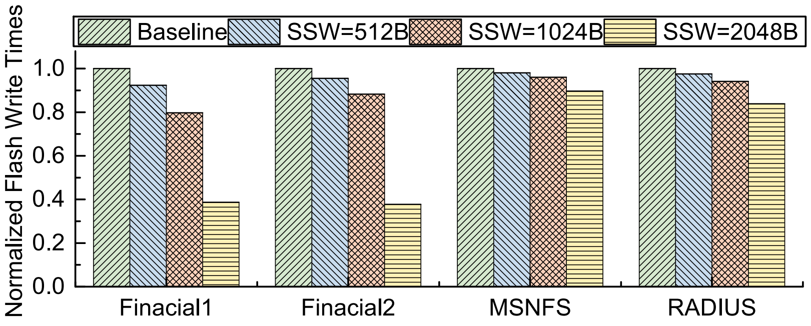


Fig. 16. The normalized flash write times.

3.3.4. Flash write times

NAND flash can only be programmed and erased within a limited number of times. This is often referred to as the maximum number of program/erase cycles (P/E cycles) it can sustain over the life of the flash memory. Frequent program and erase operations cause serious endurance problem. As shown in Fig. 16, the use of PCM decreases the write times of NAND flash significantly. Specially, in Financial1 and Financial2, the write times decrease to almost 60%. Moreover, the small write performance of PCM is much higher than NAND flash and the write latency is much shorter. The decreasing of NAND flash write times can be beneficial to the write performance as well as the P/E times.

3.3.5. Flash erase times

NAND flash memory can only be erased within a limited number of times and too many erase operations will cause serious lifespan problem. In our fusion storage system, small write data and metadata are stored in PCM, which can significantly reduce the number of NAND flash erase operations and improve the overall lifespan of NAND flash. Experimental results are shown in Fig. 17 and we find the use of PCM decreases the erase times of NAND flash significantly in some traces and not that evident in some workloads. Specially, in Financial2, with the increasing of SSW and the erase times decrease to 20%. This is because of many small repeated write requests and the employment of PCM can reduce the

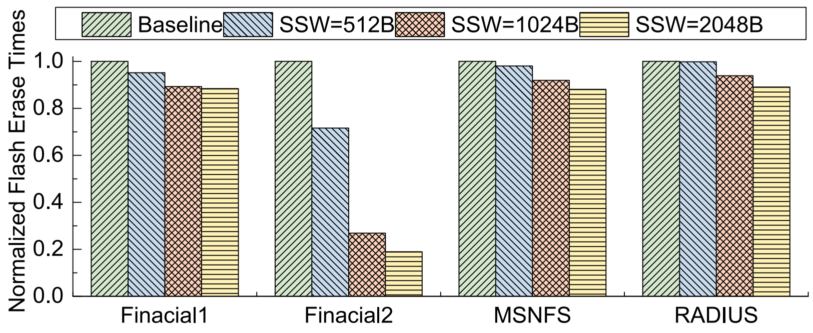


Fig. 17. The normalized flash erase times.

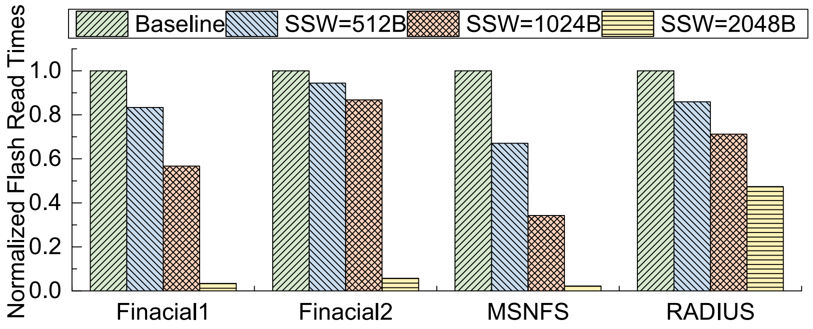


Fig. 18. The normalized flash read times.

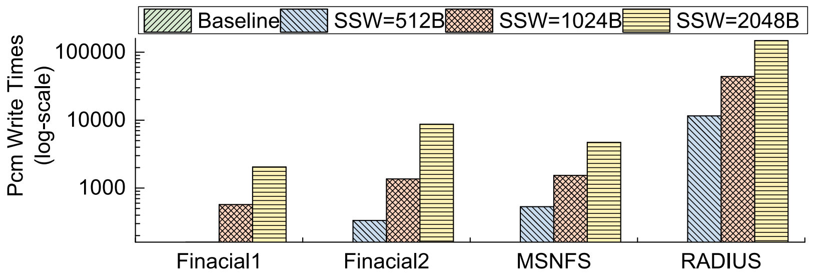


Fig. 19. PCM write times for different benchmarks.

overall operations in NAND flash. Moreover, in Financial1, MSNFS, RADIUS, the NAND flash erase times decreases 13% on average.

3.3.6. *Flash read times*

Fig. 18 shows the NAND flash read times with different *SSW*. In the figure, NAND flash read times reduce from 10%–90%. Especially in MSNFS, the read times reduce significantly, because there is no read requests in MSNFS and all read operations are resulted from WA. When *SSW* is set to 2 KB, there is almost no WA, so the read times reduce sharply. Otherwise, the size of write requests in Financial1 and Financial2 is largely 1 KB–2 KB big. If *SSW* is larger than 1 KB, 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, hence the whole performance will be improved.

3.3.7. *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 should not be ignored.

In this section, we measure the PCM read and write times in different *SSW* and the results are illustrated in Figs. 19 and 20. As show in Fig. 20, PCM read times increases with the increase of the increasing of *SSW*. Like NAND flash, the read operation of PCM is non-destructive, so it does not deliver any disadvantages on lifespan. Fig. 19 shows the write times of PCM as *SSW* varies from 512 B–2048 B. However, considering the huge endurance gap between PCM and NAND 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 by introducing PCM to deal with small read and write requests.

4. Related work

4.1. *The SSD solution*

Fusion-io puts forward a host-side SSD storage structure, as illustrated in Fig. 21(b). Compared with traditional SSD [18,31] as illustrated in Fig. 21(a), Fusion-io SSD designs FTL in the host driver and implement a new host-side FTL layer, i.e. Virtual Storage Layer (VSL). The VSL provides an address space for software 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 in the host driver program, which originally needs to be completed in the SSD firmware with powerless processor. How-

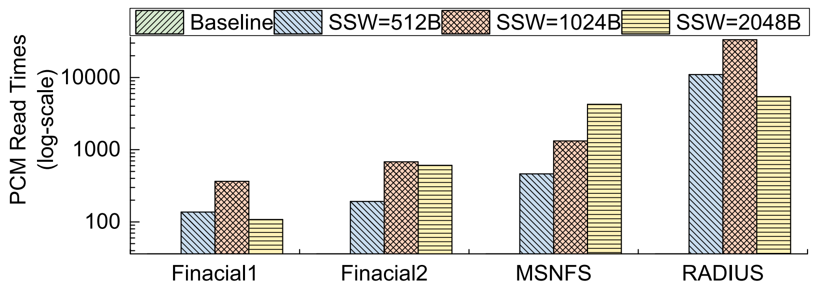


Fig. 20. PCM read times for different benchmarks.

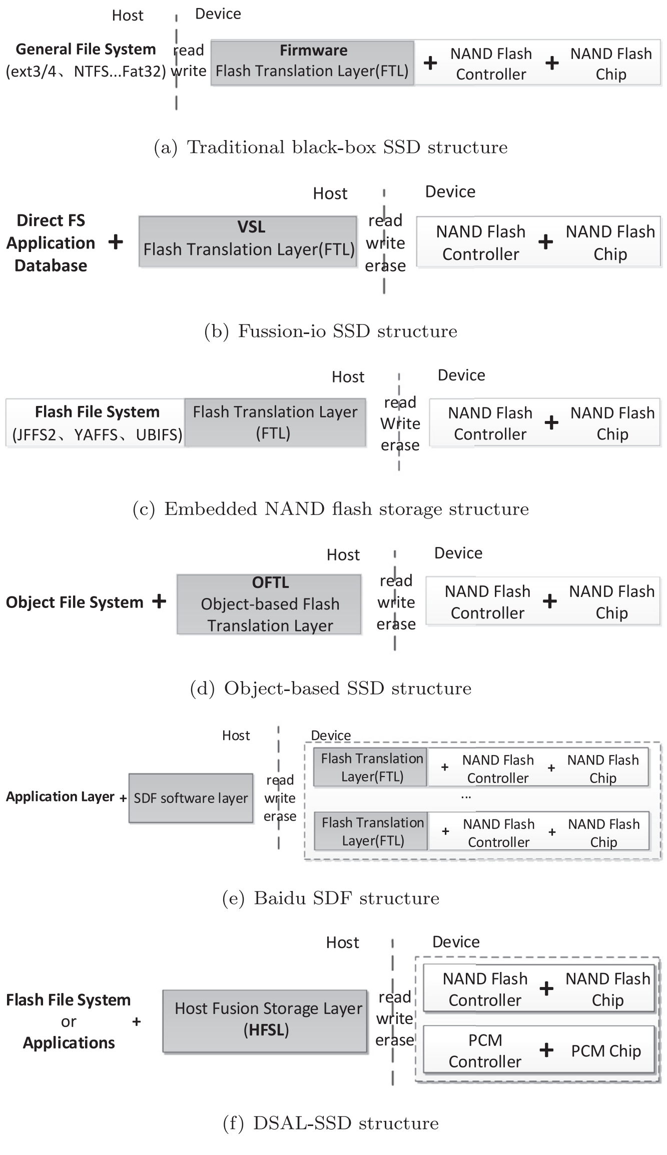

Host
Device
General File System (ext3/4、NTFS...Fat32)
read write
Firmware
Flash Translation Layer(FTL)
NAND Flash Controller
NAND Flash Chip
(a) Traditional black-box SSD structure
Direct FS Application Database
VSL
Flash Translation Layer(FTL)
read write erase
(b) Fussion-io SSD structure
Flash File System (JFFS2、YAFFS、UBIFS)
Flash Translation Layer (FTL)
(c) Embedded NAND flash storage structure
Object File System
OFTL
Object-based Flash Translation Layer
(d) Object-based SSD structure
Application Layer
SDF software layer
Flash Translation Layer(FTL)
(e) Baidu SDF structure
Flash File System or Applications
Host Fusion Storage Layer (HFSL)
PCM Controller
PCM Chip
(f) DSAL-SSD structure

Fig. 21. Structures of flash-based storage system.

ever, the Fusion-io SSD still encapsulates the SSD into a black box and users can hardly improve the system performance. Host-based FTL structure is widely used in embedded devices, as illustrated in Fig. 21(c) and server accelerator [32,33]. In addition, an object-based flash translation layer (OFTL) [11] is proposed to extend the service life and improve efficiency of SSD. OFTL merges metadata information to reduce the write amplification from the filesystem, as shown in Fig. 21(d). In ref. [34], the authors present a NAND flash system with multiple independent channels. This storage system utilizes striping, interleaving and pipelining strategies to improve the overall throughput and can significantly reduce the average service time of requests.

4.2. The applications of PCM

With the development of storage technology, a number of non-volatile memories have been developed, such as PCM, MRAM and FeRAM. These non-volatile memories obtain higher integration and lower power consumption. Among them, PCM is deemed to be one of the most promising candidates of DRAM. Compared with NAND flash, PCM has higher integration, faster read/write performance and better endurance. The most important feature is that PCM supports in place updates and can be executed in byte granularity. Most research work focuses on the study of PCM as the main memory, storage class memory and device-side buffer in embedded systems.

4.2.1. PCM-based systems

UCSD laboratory researchers develop a high performance PCM storage system named Moneta [35], which makes full use of the excellent performance of PCM. In the meanwhile, a high performance storage structure is also presented. Minerva [36] moves computation into the NVMs controllers to get closer to data. The key idea is to reduce the data travel between slow I/O interfaces (e.g. PCIe) and high-speed main memory. Minerva can make the best use of low latency and high internal bandwidth of NVMs.

4.2.2. Hybrid PCM and DRAM memory

PCM shows lower access performance with limited endurance compared with DRAM, which is avoided by conventional page replacement algorithms. Zhangling Wu introduces a new page replacement algorithm called APP-LRU to reduce the write amount to the PCM in hybrid memory [37]. The key idea of APP-LRU is to put read-intensive pages in PCM and write-intensive pages in DRAM. Soyoon Lee presents a new management called CLOCK-DWF algorithm for hybrid PCM and DRAM memory [38]. The authors use write history, not conventional read/write history to predict the write references. Moreover, it is demonstrated that leveraging frequency characteristics shows better write reduction than temporal locality. Kaimeng Chen [39] proposes a new page replacement named MHR-LRU for hybrid PCM and DRAM memory. The authors consider not only the high hit ratio but also the write burdens to PCM. Lee, Hyung Gyu [40] puts forward an energy- and performance-aware DRAM/PCM hybrid architecture and researches various DRAM/PCM hybrid configurations to significantly improve the system performance with low power consumption. G. Dhiman [41] proposes a power efficient hybrid memory system for DRAM and PCM. The authors expose the challenges of incorporating DRAM with PCM and develop an effective memory architecture to resolve these problems.

4.2.3. PCM with NAND flash

Yang Hu proposes a FTL scheme named HAT to earn better energy and performance by separating the metadata of address mapping and stored data [42]. Using PCM to store most of mapping information, the data and metadata path can be decoupled. The overall latency can be reduced by hiding the address mapping time. Duo Liu tries to avoid frequent writes in PCM when adopted to store mapping tables [43]. By eliminating redundant write, the amount of data needed to be written to PCM is reduced. Duo Liu presents a new flash management scheme named WAB-FTL to manage the NAND flash system with PCM storing the address mapping table [44]. The key idea of WAB-FTL is to delay and merge the mappingtable updates and hence the PCM write activities can be significantly reduced. L. Long [45] designs a space reuse strategy in PCM-based embedded systems and improves the overall space utilization with extremely low overhead. Guangyu Sun [46] proposes a new hybrid storage architecture for NAND flash. The key idea behind this design is to use PCM rather than NAND flash to store log data. The performance, energy consumption and lifetime of the NAND flash memory are all improved.

4.3. SDS and SDF

Software-defined storage (SDS) provides a new evolution for storage management and deployment. The key idea of SDS is to decouple the control software and the hardware it is managing. In SDS, the underlying hardware is abstracted out like storage virtualization and the software enables policies for managing the storage and data services. With SDS, storage is virtualized as a storage pool. For users, service interfaces are provided and it may allow users to build their storage system with their own hardware and provided management software. For administrators, they are free from the complex management of storage devices. They can set the policies how storage devices are managed and what services are provided.

Baidu proposes Software-Defined Flash (SDF) [17] that allows host to access flash directly, as shown in Fig. 21(e). SDF explores and exploits the advantages of NAND flash by its hardware/software co-designed architecture. SDF exposes individual flash channels to the host software and makes a lot of performance optimization according to their application environment as well as the physical properties of SSD. The software can access individual flash channels directly under user space without falling into the kernel mode. Moreover, software can define the way data organized and scheduled to make full use of the high performance of SSDs.

In detail, SDS focuses on the storage virtualization and the primary purpose is to make the storage system configurable and easily managed. Among heterogeneous block storage devices, users can manage their data on the heterogeneous storage devices without the help of conventional administrators, i.e. all management is automated.

Similar but different, our software-defined storage system aims at different aspects compared with SDS and SDF. For SDS, the devices they deployed, whatever the flash-based SSDs and PCM-based SSDs, are all still packaged into black-box module. Firmware algorithms (FTL schemes and channel management algorithms etc.) are formalist and cannot be flexibly configured according to the characteristics of applications. Moreover, the software is unclear about the statistical information of hardware, e.g. wear information and channel states. For SDF, the channel interfaces are exposed to the host software and the software can fully explore the hardware parallelism. However, FTL algorithms are still implemented in the device-side, which loses the opportunity to obtain the equipment and statistical information and earn customizable algorithms according to the characteristics of applications.

Our design implements a Host Fusion Storage Layer (HFSL) that supports flexible I/O schedule algorithms and address mapping with variable allocation size. Fig. 21(f) shows the structure of our software-defined fusion storage system. Via HFSL, the SSD exposes the number of channels, erase count and data distribution in PCM and NAND flash to the host, designs FTL algorithm closer to filesystem and manages the SSD as a user-visible structure. The host software, accessing to the PCM/Flash directly and defining the storage methods, can effectively organize the data while making full use of the performance potential of SSDs and decreasing write amplification of NAND flash.

5. Conclusion

In SSD-based storage systems, the essential metadata information and address mapping table are easy to be lost due to the volatile property of DRAM, hence being difficult to guarantee the reliability of the information. Moreover, most SSDs are encapsulated into a black box, using the NAND flash memory with a controller running FTL algorithms architecture. The black-box-modeled SSD is non-visible to host and firmware cannot obtain the equipment information and statistical information to make full use of the performance potential of NAND flash. 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 algorithms to reduce the write amplification and latency.

To address these problems, we design a user-visible solid-state storage system with software-defined fusion methods for PCM and NAND flash and implement a Host Fusion Storage Layer (HFSL) to manage the system. Our fusion storage system exposes the number of channels, erase counts and data distribution in PCM/NAND flash to the host and designs FTL algorithm closer to file system to obtain more semantic information of data accessing and manage the storage device as a visible structure. In HFSL, we implement address mapping with variable allocation size and flexible I/O scheduling algorithms to support the persistent superior performance of the system. On one hand, PCM can be used to store metadata or address mapping table information to improve the reliability. On the other hand, PCM can be used as the buffer of NAND flash to improve the small write performance. By our software-defined fusion storage structure, the system obtains benefits of lifespan extension and performance improvement. We implement a hardware prototype with actual NAND flash/PCM chips and run some classical benchmarks on our actual platform to demonstrate the effectiveness of proposed designs.

However, entire software layers need to be modified manually to make use of the proposed storage system, which delivers potential burdens and the feasibility to legacy applications. Our future work is to undergo software failures and improve the stability and reliability of proposed fusion storage system.

Acknowledgment

This work was supported by the National High Technology Research and Development Program (863 Program) No. 2015AA015301, No. 2015AA016701, No. 2013AA013203; NSFC No. 61303046, No. 61472153, No. 61173043; State Key Laboratory of Computer Architecture, No. CARCH201505; Wuhan Applied BasicResearch Project (No. 2015010101010004); This work was also supported by Key Laboratory of Information Storage System, Ministry of Education, China.

References

[1] J. Brodkin, Hurricane Sandy takes data centers offline with flooding, power outages, 2012. http://arstechnica.com/information-technology/2012/10/hurricane-sandy-takes-data-centers-offline-with-flooding-power-outages/.
[2] S. Games, Shanda Games, 2008. http://www.shandagames.com/us-en/index.html.
[3] S. Jung, Y.H. Song, Data loss recovery for power failure in flash memory storage systems, J. Syst. Archit. 61 (1) (2015) 12–27.
[4] B.C. Lee, E. Ipek, O. Mutlu, D. Burger, Architecting phase change memory as a scalable DRAM alternative, in: Proceedings of the 36th Annual International Symposium on Computer Architecture, in: ISCA '09, ACM, New York, NY, USA, 2009, pp. 2–13.
[5] Micron P5Q PCM datasheet, http://www.micron.com/.
[6] Micron P8P PCM datasheet, http://www.micron.com/.
[7] A. Akel, A.M. Caulfield, T.I. Mollov, R.K. Gupta, S. Swanson, Onyx: a prototype phase change memory storage array, HotStorage, 2011.
[8] Y. Hua, X. Liu, W. He, D. Feng, Design and implementation of holistic scheduling and efficient storage for flexray, IEEE Trans. Parallel and Distrib. Syst. 25 (10) (2014) 2529–2539.
[9] D. Zhan, Spatiotemporal capacity management for the last level caches of chip multiprocessors, Computer Engineering, University of Nebraska-Lincoln, 2012 Ph.D. Dissertation Thesis.
[10] Fusion-io, Leveraging host based flash translation layer for application acceleration, 2012. http://flashmemorysummit.com/.
[11] Y. Lu, J. Shu, W. Zheng, Extending the lifetime of flash-based storage through reducing write amplification from file systems, in: Proceedings of the 11th USENIX Conference on File and Storage Technologies, in: FAST'13, USENIX Association, Berkeley, CA, USA, 2013, pp. 257–270.
[12] Y. Cai, E.F. Haratsch, M. McCartney, K. Mai, FPGA-based solid-state drive prototyping platform, in: 2011 IEEE 19th Annual International Symposium on Field--Programmable Custom Computing Machines (FCCM), 2011, pp. 101–104.
[13] W.K. Josephson, L.A. Bongo, K. Li, D. Flynn, DFS: a file system for virtualized flash storage, Trans. Storage 6 (3) (2010) 14:1–14:25.
[14] D. Woodhouse, JFFS: the journalling flash file system, Ottawa Linux Symposium, vol. 2001, 2001.
[15] YAFFS, YAFFS: yet another flash file system, 2015. http://www.yaffs.net/.
[16] UBIFS, UBIFS: unsorted block image file system, 2015. http://git.infradead.org/linux-ubifs.git.
[17] J. Ouyang, S. Lin, S. Jiang, Z. Hou, Y. Wang, Y. Wang, SDF: software-defined flash for web-scale internet storage systems, in: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems, in: ASPLOS '14, ACM, New York, NY, USA, 2014, pp. 471–484.
[18] Micron, Micron SSDS, 2015. http://www.micron.com/solutions/client-ssd-storage.
[19] S. Swanson, A. Caulfield, Refactor, reduce, recycle: restructuring the I/O stack for the future of storage, Computer 46 (8) (2013) 52–59.
[20] B. Hoefflinger, ITRS 2028 international roadmap of semiconductors, in: B. Hoefflinger (Ed.), CHIPS 2020 VOL. 2, Springer International Publishing, 2016, pp. 143–148.
[21] P. Zhou, B. Zhao, J. Yang, Y. Zhang, A durable and energy efficient main memory using phase change memory technology, in: Proceedings of the 36th Annual International Symposium on Computer Architecture, in: ISCA '09, ACM, New York, NY, USA, 2009, pp. 14–23.
[22] M.K. Qureshi, V. Srinivasan, J.A. Rivers, Scalable high performance main memory system using phase-change memory technology, in: Proceedings of the 36th Annual International Symposium on Computer Architecture, in: ISCA '09, ACM, New York, NY, USA, 2009, pp. 24–33.
[23] S. Raoux, G.W. Burr, M.J. Breitwisch, C.T. Rettner, Y.-C. Chen, R.M. Shelby, M. Salinga, D. Krebs, S.-H. Chen, H.-L. Lung, et al., Phase-change random access memory: a scalable technology, IBM J. Res. Dev. 52 (4.5) (2008) 465–479.
[24] F. Xia, D.-J. Jiang, J. Xiong, N.-H. Sun, A survey of phase vhange memory systems, J. Comput. Sci. Technol. 30 (1) (2015) 121–144.
[25] W. Mao, J. Liu, W. Tong, D. Feng, Z. Li, W. Zhou, S. Zhang, A review of storage technology research based on phase change memory, Chin. J. Comput. 38 (5) (2015) 944–960.
[26] Samsung K9XXG08UXA NAND flash datasheet, http://www.samsung.com/semiconductor/products/flash-storage/.
[27] A. Gupta, Y. Kim, B. Urgaonkar, 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, pp. 229–240.
[28] S. Park, Y. Kim, B. Urgaonkar, J. Lee, E. Seo, A comprehensive study of energy efficiency and performance of flash-based SSD, J. Syst. Archit. 57 (4) (2011) 354–365.
[29] UMass, UMass trace repository, 2015. http://traces.cs.umass.edu/index.php.
[30] SNIA, SNIA, block I/O traces, 2015. http://iotta.snia.org/tracetypes/3.
[31] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, H.-J. Song, A survey of flash translation layer, J. Syst. Archit. 55 (5–6) (2009) 332–343.
[32] Y.-H. Chang, P.-L. Wu, T.-W. Kuo, S.-H. Hung, An adaptive file-system-oriented FTL mechanism for flash-memory storage systems, ACM Trans. Embedded Comput. Syst. (TECS) 11 (1) (2012) 1–19.
[33] J. Kim, H. Shim, S.-Y. Park, S. Maeng, J.-S. Kim, Flashlight: a lightweight flash file system for embedded systems, ACM Trans. Embedded Comput. Syst. (TECS) 11S (1) (2012) 1–23.
[34] J.-U. Kang, J.-S. Kim, C. Park, H. Park, J. Lee, A multi-channel architecture for high-performance NAND flash-based storage system, J. Syst. Archit. 53 (9) (2007) 644–658.
[35] A.M. Caulfield, A. De, J. Coburn, T.I. Mollow, R.K. Gupta, S. Swanson, Moneta: a high-performance storage array architecture for next-generation, non-volatile memories, in: Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, 2010, pp. 385–395.
[36] A. De, M. Gokhale, R. Gupta, S. Swanson, Minerva: accelerating data analysis in next-generation SSDs, in: 2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2013, pp. 9–16.
[37] Z. Wu, P. Jin, C. Yang, L. Yue, APP-LRU: a new page replacement method for PCM/DRAM-based hybrid memory systems, in: Network and Parallel Computing, Springer, 2014, pp. 84–95.
[38] S. Lee, H. Bahn, S.H. Noh, CLOCK-DWF: a write-history-aware page replacement algorithm for hybrid PCM and DRAM memory architectures, IEEE Trans. Comput. 63 (9) (2014) 2187–2200.
[39] K. Chen, P. Jin, L. Yue, A novel page replacement algorithm for the hybrid memory architecture involving PCM and DRAM, in: Network and Parallel Computing, Springer, 2014, pp. 108–119.
[40] H.G. Lee, S. Baek, C. Nicopoulos, J. Kim, An energy- and performance-aware DRAM cache architecture for hybrid DRAM/PCM main memory systems, in: 2011 IEEE 29th International Conference on Computer Design (ICCD), 2011, pp. 381–387.
[41] G. Dhiman, R. Ayoub, T. Rosing, PDRAM: a hybrid PRAM and DRAM main memory system, in: 46th ACM/IEEE Design Automation Conference, 2009, pp. 664–669.
[42] Y. Hu, H. Jiang, D. Feng, L. Tian, S. Zhang, J. Liu, W. Tong, Y. Qin, L. Wang, Achieving page-mapping FTL performance at block-mapping FTL cost by hiding address translation, in: 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), IEEE, 2010, pp. 1–12.
[43] D. Liu, T. Wang, Y. Wang, Z. Qin, Z. Shao, PCM-FTL: a write-activity-aware NAND flash memory management scheme for PCM-based embedded systems, in: 2011 IEEE 32nd Real-Time Systems Symposium (RTSS), IEEE, 2011, pp. 357–366.
[44] D. Liu, T. Wang, Y. Wang, Z. Qin, Z. Shao, A block-level flash memory management scheme for reducing write activities in PCM-based embedded systems, in: Proceedings of the Conference on Design, Automation and Test in Europe, EDA Consortium, 2012, pp. 1447–1450.
[45] L. Long, D. Liu, J. Hu, S. Gu, Q. Zhuge, E.H.-M. Sha, A space allocation and reuse strategy for PCM-based embedded systems, J. Syst. Archit. 60 (8) (2014) 655–667.
[46] G. Sun, Y. Joo, Y. Chen, Y. Chen, Y. Xie, A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement, in: Emerging Memory Technologies, Springer, 2014, pp. 51–77.

Zheng Li received the BE degree in computer science and technology from the Huazhong University of Science and Technology (HUST), China, in 2013. He is currently working toward the PhD degree in computer architecture from HUST. His research interest includes reconfigurable computing on FPGA and non-volatile memory-based storage system. He publishes several papers in major conferences including NVMSA, DATE, ICPP etc..

Fang Wang received the BE, ME and PhD degrees in computer science and technology from the Huazhong University of Science and Technology (HUST), China, in 1994, 1997 and 2001, respectively. She is a professor of the School of Computer Science and Technology, HUST. Her research interests include computer architecture, massive storage systems and parallel file systems. She has more than 40 publications to her credit in journals and international conferences including ACM TACO, SC, MSST, ICPP, ICA3PP, HPDC and ICDCS.

Jingning Liu received the BE degree in computer science and technology from the Huazhong University of Science and Technology (HUST), China, in 1982. She is a professor in the HUST and engaged in researching and teaching of computer system architecture. Her research interests include computer storage network system, high-speed interface and channel technology, embedded system and FPGA design. She has over 20 publications in journals and international conferences including ACM TACO, NAS, MSST and ICA3PP.

Dan Feng received the BE, ME and PhD degrees in computer science and technology from the Huazhong University of Science and Technology (HUST), China, in 1991, 1994 and 1997, respectively. She is a professor and vice dean of the School of Computer Science and Technology, HUST. Her research interests include computer architecture, massive storage systems and parallel file systems. She has more than 80 publications to her credit in journals and international conferences, including IEEE Transactions on Parallel and Distributed Systems (TPDS), JCST, USENIX ATC, FAST, ICDCS, HPDC, SC, ICS and ICPP. She is a member of the IEEE.

Yu Hua received the BE and PhD degrees in computer science from the Wuhan University,China, in 2001 and 2005, respectively. He is a professor at the Huazhong University of Science and Technology, China. His research interests include computer architecture, cloud computing and network storage. He has more than 60 papers to his credit in major journals and international conferences including IEEE Transactions on Computers (TC), IEEE Transactions on Parallel and Distributed Systems (TPDS), USENIX ATC, FAST, INFOCOM, SC, ICDCS and MSST. He has served on the program committee for multiple international conferences, such as INFOCOM, RTSS, ICDCS, MSST, ICNP, ICPP, IWQoS. He is a senior member of IEEE and CCF.

Wei Tong received the BE, ME and PhD degrees in computer science and technology from the Huazhong University of Science and Technology (HUST), China, in 1999, 2002 and 2011, respectively. She is a lecturer of the School of Computer Science and Technology, HUST. Her research interests include computer architecture, network storage system and solid state storage system. She has more than 10 publications in journals and international conferences including ACM TACO, MSST, NAS, FGCN.

Shuangwu Zhang received the BE and ME degrees in computer science and technology from the Huazhong University of Science and Technology (HUST), China, in 2012 and 2015, respectively. His research interest includes computer architecture and storage systems.