Rethinking the File System Design on Flash-based Storage

Tsinghua University ▌Youyou Lu · Jiacheng Zhang · Jiwu Shu

Flash memory is gaining great popularity in both embedded and enterprise systems. Compared to legacy hard drives, flash devices get rid of mechanical components and significantly improve performance. However, flash memory is not merely a faster storage media, and its characteristics provide opportunities of new storage system designs. In this paper, we summarize disruptive techniques on flash storage respectively in areas of storage architecture, system software and distributed protocols. With a focus on file system designs for flash storage, we describe our work from three aspects: the hardware support, the storage management and the namespace management. With preliminary research in this area, we believe that it needs disruptive techniques in I/O stack to exploit the potentials of flash memory.

1. Introduction

Computing, storage, networking are the three components in computer systems. Among the three components, storage is the only one that contains mechanical parts, i.e., the moving parts in the rotating hard disk drives (HDDs). The mechanical moving parts in HDDs limit the performance of storage devices in terms of both access bandwidth and latency [10].

In recent years, flash memory, an electronic storage media, has been greatly developed. While flash memory is employed in storage systems, it breaks the performance limit of secondary storage. The I/O latency is reduced from milliseconds to tens of microseconds, and the read/write bandwidth is improved from megabytes per second (in SATA form) to gigabytes per second (in PCIe form) [5]. The adoption of flash memory in the storage systems enables significantly better I/O performance in the computer system.

In addition to the performance difference, flash memory also has characteristics that are quite different from those of magnetic disks. Random I/Os, which are the costly operations in HDDs, are much faster in flash storage [11]. Optimizations for random I/Os are not that necessary in flash storage. In flash storage, a cell can only be programmed once, and it needs an erase before being overwritten [6]. This is known as the no-overwrite property. Due to this property, updates are performed in new places while leaving the old ones for garbage collection. Since both new and old versions are kept, write consistency in flash storage can be easily supported. Flash memory also has other advantages, e.g., low energy consumption [7] and light weight, which can make the storage system design different.

Meanwhile, flash memory has several weaknesses compared to HDDs. One difference is that flash memory has endurance problem. A flash memory cell can endure limited number of program/erase (P/E) cycles [6, 15, 22]. To extend the lifetime of flash storage, write traffic needs to be controlled. Unbalanced read and write cost is another property that makes flash memory different from HDDs [24]. Flash memory has high cost in random writes, but low cost in random reads. In addition, updates are performed to new locations while leaving the old versions for garbage collection, due to the no-overwrite property. The garbage collection could cause unexpected performance degradation [6]. Storage systems need to take the shortages of flash memory into consideration, so as to achieve the consistent benefits.

Flash-based solid state drives (SSDs) exports the same storage API as the HDDs. As shown in Figure 1, the SSD form makes flash memory easily adopted in current storage systems without changing current I/O stack. However, this form also prevents the flash memory properties from being fully exploited, and leaves some disadvantages without being addressed. In this paper, we discuss the I/O stack implications of flash memory with a focus on the file system design.

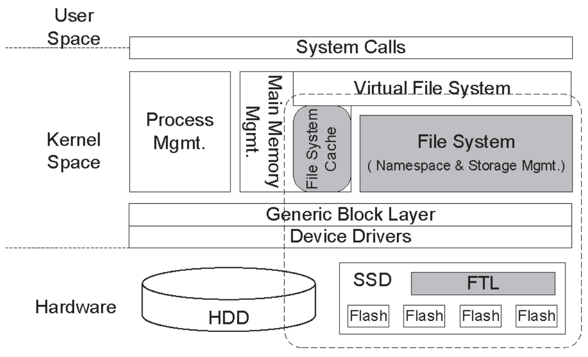


Fig. 1 File systems on HDDs and SSDs

The rest of the paper is organized as follows. Section 2 summarizes disruptive techniques of I/O stack in areas of storage architecture, system software and distributed protocols, and analyzes the problems in file systems on flash storage. Section 3, 4 and 5 respectively present our research work on local flash file system in different layers: the hardware support, the storage management and the namespace management. Section 6 further discusses the open challenges, and Section 7 concludes the paper.

2. Storage System Implications of Flash Memory

2.1 Implications in the I/O Stack

Solid state drive (SSD) is a popular form of flash storage nowadays. To hide the hardware details of flash memory from the software, SSD uses a flash translation layer (FTL) to emulate the same read and write interface as in HDDs. This form provides good compatibility with legacy HDDs, but prevents the hardware from sensing the real intent of software. This leads to unnecessary operations in SSDs and hurts system performance. One typical example is the deletion operation in SSD-based storage systems. The file system deletes a data file by invalidating its metadata. The SSD does not know the deletion of the file data and moves them during garbage collection, which causes necessary move operations. The trim operation is introduced to bridge this gap [4]. Some other operations, like *exist* and *ptrim* [25], are also proposed to make better use of flash memory in the SSD form. While the introduced new operations make better use of flash storage, gains are limited.

Recently, researchers are realizing that the I/O stack needs a disruptive change to better exploit the potentials of flash memory. Representative works include the changes in the storage architecture, the system software and the distributed protocols.

2.1.1 Changes in Storage Architecture

Hardware Interface. In a flash storage device, flash memory cells are accessed in parallel from different planes, chips, channels. This *internal parallelism* property of flash devices provides scalable internal I/O bandwidth [12, 16]. Traditional SATA, SCSI or SAS interface becomes the bandwidth bottleneck. Thus, PCIe interface is proposed to improve the external storage bandwidth, as in FusionIO [3].

Hardware Notification. Interruption is a common hardware notification mechanism in HDD-based storage systems. The mechanism allows CPU to continue execution without waiting for I/Os. However, flash memory has much lower I/O latency. Recent research argues that poll is better than interruption in flash-based storage systems [30].

Hardware-assisted Storage Consistency. The no-overwrite property of flash memory naturally keeps both the old and new versions of a data page. Transaction recovery protocols for storage consistency are now moved from software to hardware, in order to either lower the overhead by leveraging the no-overwrite property [28, 27, 20, 23, 19] or improve performance by utilizing the internal bandwidth [13]. Our work LightTx is a high-performance and lightweight transaction protocol in SSDs to support storage consistency. We will present LightTx in Section 3.

2.1.2 Changes in System Software

Storage Management. System software, including both file systems and database management systems, is redesigned or revised to use the new storage hardware. New file systems have been designed from scratch for flash storage. Direct File System (DFS) [17] is a file system which is built on top of FusionIO's Virtual Storage Layer (VSL), a kind of software FTL. DFS leverages VSL for storage allocation and thus eliminates the duplicated storage allocation functions, which appear in both the file system and the FTL in legacy storage systems. Similar idea has also been proposed in Nameless Write technique [31], which offloads the storage allocation function from file systems to storage devices by extending the storage interface. F2FS [18] is another newly developed file system for flash storage. F2FS uses the log-structured data organization and is optimized for flash device organization. These optimizations enable better data performance in file systems.

Legacy file systems also make some changes to better use flash memory, including new allocation policies (e.g., in Btrfs [1]), trim operation support (e.g., in Ext4 [2]), and data grouping algorithms (e.g., in SFS [24]).

We propose an objected-based flash storage system named OFSS to manage storage space in objects [22]. In OFSS, storage management functions in both file systems and FTLs are combined with two goals. One is to deduplicate the redundancy functions, and the other is to enable the opportunity of the co-design between hardware and software. We will present OFSS in Section 4.

Namespace Management. Directory tree is the dominate form in file system namespace management for over forty years. Directory tree metadata introduces high overhead due to its random and frequent update pattern. With the new flash storage media, we start to rethink the namespace management for flash storage and propose a new file system named ReconFS [21]. ReconFS is designed based on the observation that flash memory has unbalanced read and write cost. ReconFS decouples the maintenance of the volatile and the persistent directory trees, so as to reduce the write overhead. Directory tree metadata is persisted in different ways to meet the consistency and persistence requirements of namespace management. The high read performance enables quick reconstruction of directory trees after system crashes. With the new namespace management in ReconFS, metadata performance is improved. We will present ReconFS in Section 5.

2.1.3 Changes in Distributed Protocols

In distributed systems, distributed protocols can have new designs on flash storage. Researchers from Microsoft propose CORFU [8], which puts distributed flash storage into a single pool and provides the log-structured accesses. Tango [9] further develops the replicated data structure based on the shared log provided in CORFU. By leveraging the log-structured update pattern in flash storage, they provide strong consistency among different machines without complex network protocols.

In distributed file systems, data migration is commonly used for load balance. However, migration also introduces extra writes, which hurt the endurance of flash storage. EDM is an endurance-aware data migration that is designed for load balance in distributed file systems [26].

2.2 Implications in the File System Design

Improper Designs in Legacy File Systems on Flash Storage. Traditional file systems are designed and optimized for HDDs. In storage systems based on the commonly used solid state drives (SSDs), traditional file systems cannot fully exploit the advantages of flash memory, and even hurt its performance or endurance. We summarize the key problems between traditional file systems and SSDs as follows.

- **Missing Consideration of the Endurance Problem.** A flash memory cell wears when it is programmed and erased. Flash memory endurance is sensitive to write traffic. While HDDs do not have endurance problem, traditional file systems have not taken endurance into consideration, which however is important in flash storage systems. Therefore, file systems for flash storage need to consider the endurance problem, as well as the reliability and performance issues.
- **Opportunity Loss.** Traditional file systems use journals or shadow pages to maintain multi-version data in the software level. The maintenance doubles the write size and introduces synchronized writes, which degrade the system performance and shorten flash device lifetime. However, the flash property of no-overwrite can be easily used to provide multi-version data and decrease the overhead of transactions. Also, each flash memory page has a metadata area, called the OOB (out-of-band) area, which can provide atomic access together with page data. The reserved space can be used to simplify the metadata management of storage system and the consistency.
- **Function Redundancy.** One function of traditional file systems is to manage the storage space. However, flash translation layer (FTL) in SSDs also needs to manange the storage space, due to the no-overwrite property and the limited erase cycles. These redundant functions with traditional file systems not only introduce access overhead, but also prevents file systems from sensing and exploiting the characteristics of flash memory. The flash-based file systems are suggested to co-design with flash memory for better performance and longer lifetime.
- **Optimization Mismatch.** Traditional file systems are optimized based on HDDs' properties, such as sequential access. Flash storage has unbalanced performance for read and write where read operations are usually faster than write operations. Random

access to flash storage not only decreases the system performance but also accelerate the wear process, leading to shorter lifetime. Flash-based storage systems may take properties of flash memory to design better ways for metadata and data management.

In addition, other properties of flash devices, such as internal parallelism, block-level erase, also have big effect on file system design.

Our Work. Legacy file systems which are heavily designed for HDDs fail to fully exploit the advantages of SSDs. Therefore, we start the research on file system designs that are suitable for flash storage. Our work starts from three layers: the hardware support, the storage management, and the namespace management. As shown in Figure 2, our work LightTx, OFSS and ReconFS are proposed in each of the three layers. The three works are presented in the following three sections.

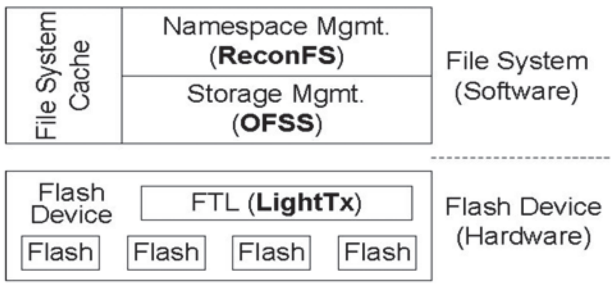


Fig. 2 Our Work in Different Layers: LightTx (Section 3), OFSS (Section 4), ReconFS (Section 5)

3 Hardware Support: LightTx

Journaling is a commonly used transaction mechanism in file systems to provide storage consistency. In journaling, data are write twice, and synchronizations are required to keep the write sequence. Maintaining storage consistency in system software is costly. The no-overwrite property of flash memory provides new opportunities of low- overhead data versioning. Based on this, we propose LightTx, a high-performance and lightweight transaction protocol with hardware support, to efficiently support storage consistency [20, 19].

For decades, transactions have been widely used in database management systems, file systems, and applications to provide the ACID properties, but usually at the cost of implementation complexity and degraded performance. Flash memory properties of no-overwrite and high random I/O performance (comparatively to hard disks) favor the shadow paging approach [14]. A page is atomically updated in SSDs by simply updating the mapping entry in the FTL mapping table. Because of this simplicity provided by the FTL mapping table, supporting transactions inside SSDs is attractive.

Recent researches [27, 28] propose to support transactions inside an SSD by introducing a new interface, WriteAtomic, and providing multi-page update atomicity. Transaction support in the SSD nearly doubles system performance due to the elimination of duplicated log writes [27, 28]. Unfortunately, these proposals support a limited set of isolation levels, which makes the system inflexible and hurts internal parallelism. On the other hand, flexible transaction support, transaction aborts and the need for fast recovery lead to high overhead.

We design a new embedded commit protocol, LightTx, to support flexible isolation levels in the system with atomicity and durability guarantees provided inside the SSD, while achieving low hardware overhead (in terms of garbage collection and mapping persistence) for tracking transactions states. To enable such a mechanism, we make the following specific contributions:

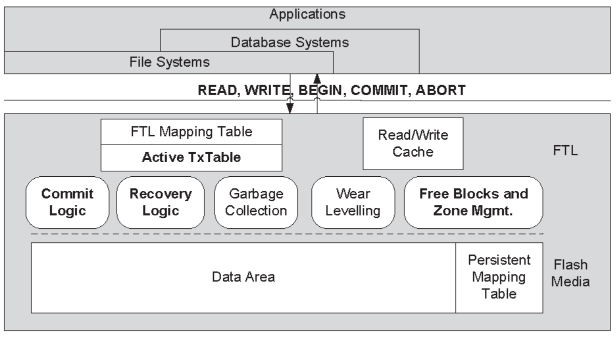
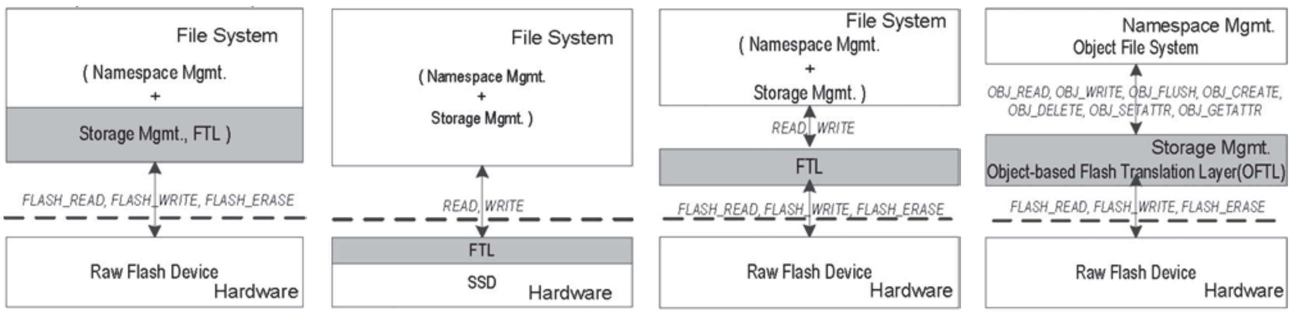


Fig. 3 The LightTx Architecture

- We extend the SSD interface with transaction semantics to support transaction recovery in hardware.
- We introduce a page-independent commit protocol to support simultaneous update of multiple versions of a page concurrently written by different transactions. This protocol provides flexible isolation level choices to the system.
- We design a new zone-based transaction state tracking scheme that tracks the live transactions and periodically identifies and retires the dead transactions. This reduces transaction state tracking cost, making our proposal a lightweight design.



(a) FS-Embedded FTL (b) SSD-Embedded FTL (c) Software FTL (d) Object-based FTL

Fig. 4. Evolution of the FTL Architecture.

Our evaluation of LightTx using database traces shows a transaction throughput increase of up to 20.6% with relaxed isolation levels compared to strict isolation. The overhead of LightTx is nearly the lowest in both garbage collection and mapping persistence time compared to existing embedded transaction designs [27, 28].

Transaction Recovery in Hardware. LightTx extends the FTL functions in SSDs to support transaction atomicity and durability. As shown in Figure 3, in addition to the common modules, LightTx introduces three new modules (the active TxTable, commit logic, and recovery logic) and revises the free block management using a zone-based scheme. The commit logic extracts the transaction information from the extended transactional interface and tracks the active transactions using the active TxTable. The active TxTable is a list of active transaction lists, and each active transaction list links the page metadata of all pages of the transaction. Each entry in the active TxTable records the mapping from a page's address to its physical address. The recovery logic differentiates the committed transactions from the uncommitted and redoes the committed ones during recovery. In order to support the transaction information exchange between the system and the device, we add the BEGIN, COMMIT, and ABORT commands, which are similar to the transaction primitives used in the system, and extend the WRITE command.

Page-Independent Commit Protocol. In the commit protocol design, LightTx aims to minimize dependences between different pages and different versions of the same page. To achieve this goal, LightTx limits the operation of commit logic within each transaction, as each page can be identified whether it belongs to some transaction by storing the transaction identifier in the page metadata. Also, LightTx delays the FTL mapping table updates until the commit/abort of each transaction instead of doing updates on a page-by-page basis to reduce conflicts. Transaction versions, as opposed to page versions, are used to determine the update sequence of the FTL mapping table. These allow concurrent execution of different transactions even with overlapped updates.

Zone-based Transaction State Tracking Scheme. Since pages of each transaction may be scattered across many different flash blocks, state tracking can become costly for flexible isolation levels. To improve efficiency, zone-based transaction state tracking scheme is used to reduce the tracking overhead in two aspects. First, we track transaction states by tracking the states of flash blocks instead of flash pages. Second, we reduce the number of transactions to be tracked by keeping the live transactions separated from the dead, which is achieved by classifying flash blocks into different zones and tracking them separately in these zones.

4 Storage Management: OFSS

Storage management in file systems has duplicated functions with FTLs in SSDs. In the storage management aspect, we propose an object-based flash storage system, OFSS, to improve the write efficiency in flash storage systems [22, 29].

The endurance problem is a major concern in flash storage. This problem is getting worse as storage density increases with the adoption of multi-level cells. Prior work has addressed wear leveling and data reduction, but there is significantly less work on using the file system to improve flash lifetimes. Some common mechanisms in traditional file systems, such as journaling, metadata synchronization, and page-aligned update, can induce extra write operations and aggravate the wear of flash memory. This problem is called *write amplification from file systems*. As far as we know, we are the first to identify the write amplification problem in the file system level.

In order to mitigate the write amplification problem, we introduce an object-based flash translation layer design (OFTL), in which mechanisms are co-designed with flash memory. By leveraging page metadata, OFTL enables lazy persistence of index metadata and eliminates journals while keeping consistency. Coarse-grained block state maintenance reduces persistent free space management overhead. With byte-unit access interfaces, OFTL is able to compact and co-locate the small updates with metadata to further reduce updates. Experiments show that an OFTL-based system, OFSS, offers a write amplification reduction of 47.4% 89.4% in SYNC mode and 19.8% 64.0% in ASYNC mode compared with Ext3, Ext2, and Btrfs on an up-to-date page-level FTL.

Object-based Flash Translation Layer (OFTL). In embedded systems, flash is directly managed by the file system, where the mapping, garbage collection and wear leveling are implemented, known as the *FS-embedded FTL* in Figure 4 (a). With the increased capacity and reduced price, flash devices are adopted in enterprise computer systems. FTLs are implemented in the firmware to provide the same block interface as HDDs. This is known as the *SSD-embedded* FTL in Figure 4 (b). As the SSD-embedded FTL with limited computing and memory resources is hard to meet the requirements in SSDs with increased capacity, FusionIO introduces the *software FTL* which implements FTLs in the software, sharing the host computing and main memory resources. Figure 4 (c) shows the architecture. Although software FTLs have shown better performance than embedded FTLs, the narrow block interfaces prevent optimizations from either the file system or the FTL. File semantics are hidden behind the narrow interface, and flash memory properties are opaque to the file system. We propose an new FTL architecture, the object-based *FTL (OFTL)*, for better cooperation with the file system and flash memory, as shown in Figure 4 (d).

The OFTL-based architecture offloads storage space management from the file system to the OFTL for better co-designs of the file system and the FTL. OFTL accesses the raw flash device in page-unit interfaces, while exporting byte-unit access interfaces to the file system. And thus, OFTL translates the mapping from the logical offset of each object to the flash page address.

System Co-design with Flash Memory. The OTFL uses three techniques to leverage the characteristics of the underlying flash device.

- **Backpointer-Assisted Lazy Indexing.** The lazy indexing technique, in which the type specific backpointer is employed in the page metadata of each indexed page for lazy persistence of the index metadata, is developed to reduce the frequency of index persistence while maintaining consistency.
  To reduce the scan time of inverse index to reconstruct the index metadata after system failures, we use the updating window to track the recently allocated flash blocks that have not completed the index metadata persistence. The updating window is maintained by the checkpoint process, which is triggered when the number of free pages in the updating window drops below a threshold and a window extending is needed. After a crash, the recovery process needs to only read the blocks whose addresses are in the updating window metadata and check whether they are referenced by the index.
- **Coarse-Grained Block State Maintenance.** In flash memory, each page has three states: FREE, VALID, and INVALID. Since the pages are written sequentially inside a flash block, the latest allocated page number is used to separate the free pages from the inuse for the updating block. Therefore, OFTL differentiates between free and inuse pages by tracking the flash block states, and free space management cost is reduced with the state maintenance in flash block units instead of page units. OFTL further reduces the cost by bringing down the frequency of metadata persistence.The state persistence is relaxed with the proper conditions satisfied. In summary, free space management benefits from the coarse-grained block state maintenance because of the reduced metadata cost both from the flash block granularity state tracking and the reduced persistence of states.
- **Compacted Update.** With byte-unit access interfaces, OFTL is able to identify partial page updates, which update only part of one page, both for the small updates less than one page and the heads/tails of large updates. The compacted update technique compacts the partial page updates of the same object and co-locates them with its object metadata page to reduce the number of update pages.

While partial page updates are absorbed in differential pages, which are indexed in the diff-layout, full page updates are directly written to object pages. Object pages are indexed in the layout, which is the collection of all extents of one object that keep the start address and length pairs of full pages. The layout is recorded in the object metadata page.

5 Namespace Management: ReconFS

Directory tree has been used for file system namespace management for forty years. Namespace metadata causes high overhead due to its random and frequent writes, which remains a challenge in HDD-based storage systems. With the property of unbalanced read and write cost of flash memory, we introduce a novel write-back mechanism in namespace management, and the new file system ReconFS dramatically reduces the write overhead [21].

Namespace metadata are intensively written back to persistent storage due to system consistency or persistence guarantees. Since the no-overwrite property of flash memory requires writes to be updated in free pages, frequent writeback introduces a large dynamic update size. Even worse, a single file system operation may scatter updates to different metadata pages (e.g., the create operation writes both the inode and the directory entry), and the average update size to each metadata page is far less than one page size (e.g., an inode in Ext2 has the size of 128 bytes). A whole page needs to be written even though only a small part in the page is updated. Endurance, as well as performance, of flash storage systems are affected by namespace metadata accesses due to frequent and scattered small write patterns.

To address these problems, we propose a reconstructable file system, ReconFS, which provides a volatile hierarchical namespace and relaxes the writeback requirements. Due to the page unit access and no-overwrite property, the frequently updated metadata brings large write amplification

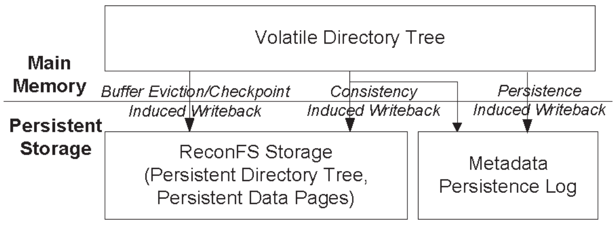

Volatile Directory Tree
Main Memory
Buffer Eviction/Checkpoint Induced Writeback
Consistency Induced Writeback
Persistence Induced Writeback
Persistent Storage
ReconFS Storage (Persistent Directory Tree, Persistent Data Pages)
Metadata Persistence Log

Fig. 5 Decoupled Maintenance of the Volatile and Persistent Directory Trees (ReconFS Framework)

to the flash devices. We decouple the maintenance of the volatile and persistent directory trees, so as to reduce the high overhead of maintaining a persistent directory tree. Namespace consistency and persistence concerns arise when the writeback of persistent directory tree is relaxed. ReconFS ensures the two properties respectively using the *embedded connectivity* and the *metadata persistence logging* techniques. Once system crashes, metadata that is persisted using the two techniques is read to reconstruct the directory tree. Due to the high read performance of flash devices, reconstruction of the directory tree is fast.

We implement ReconFS based on Ext2 and evaluate it against different file systems, including Ext2, Ext3, Btrfs and F2FS. Results show an up to 46.3% performance increase and 27.1% endurance improvement compared to Ext2, a file system with low metadata overhead.

Decoupled Maintenance of the Volatile and Persistent Directory Trees. ReconFS decouples the maintenance of the volatile and persistent directory trees. ReconFS emulates a volatile directory tree in main memory to provide the hierarchical namespace access. Metadata pages are updated to the volatile directory tree without being written back to the persistent directory tree.

As shown in Figure 5, ReconFS is composed of three parts: the Volatile Directory Tree, the ReconFS Storage, and the Metadata Persistence Log. The Volatile Directory Tree manages namespace metadata pages in main memory to provide hierarchical namespace access. The ReconFS Storage is the persistent storage for ReconFS file system. It stores both the data and metadata, including the persistent directory tree, of the file system. The Metadata Persistence Log is a continuously allocated space in the persistent storage which is mainly used for the metadata persistence.

Embedded Connectivity. Namespace consistency is one of the reasons why namespace metadata needs frequent writeback to persistent storage. In the normal indexing of a directory tree, the pointer and the pointed page of each link should be written back atomically for namespace consistency in each metadata operation. This not only requires the two pages to be updated but also demands journaling or ordered update for consistency. Instead, ReconFS provides namespace consistency using inverted indexing, which embeds the inverted index with the indexed data. Since the pointer is embedded with the pointed page, the consistency can be easily achieved. As well as the journal writes, the pointer updates are eliminated. In this way, the embedded connectivity lowers the frequency of metadata writeback and ensures the metadata consistency.

Metadata Persistence Logging. Metadata persistence causes frequent metadata writeback. The scattered small update pattern of the writeback amplifies the metadata writes, which are written back in the unit of pages. Instead of using static compacting, ReconFS dynamically compacts the metadata updates and writes them to the metadata persistence log. While static compacting requires the metadata updates written back to their home locations, dynamic compacting is able to cluster the small updates in a compact form. Dynamic compacting only writes the dirty parts rather than the whole pages, so as to reduce write size.

In metadata persistence logging, writeback is triggered when persistence is needed, e.g., explicit synchronization or the wake up of pdflush daemon. The metadata persistence logging mechanism keeps track of the dirty parts of each metadata page in main memory and compacts those parts into the logs. Multi-page update atomicity is simply achieved using a flag bit in each page.

ReconFS Reconstruction. During normal shutdowns, the volatile directory tree writes the checkpoint to the persistent directory tree in persistent storage, which is simply read into main memory to reconstruct the volatile directory tree for the next system start. But once the system crashes, ReconFS needs to reconstruct the volatile directory tree using the metadata recorded by the embedded connectivity and the metadata persistence logging mechanisms. ReconFS only needs to update the directory tree by scanning the unindexed zone and the metadata persistence log.

6. Open Challenges

We rethink the co-design between the software (i.e., file system) and the hardware (i.e., flash storage), and perform the research from three aspects, i.e., the hardware support, the storage management and the namespace management. File systems for flash storage still need more explorations, e.g., data layout that are friendly to flash read/write/erase operations, the file system cache management, etc. In addition, flash devices can be used in novel ways (e.g., in the memory level, in the big data platform, in the distributed systems), and flash file systems have new challenges in new environments.

1. **Memory-Level File System Design.** NAND flash has narrowed the performance gap between storage and memory. Storage devices are connected closer to the CPU. The interface of Non-Volatile DIMMs (NVDIMMs) enables storage being directly accessed from the CPU. This further reduces I/O latency and leads to high IOPS. In this case, software overhead becomes much significant as the proportion of storage media latency is dramatically reduced. Therefore, file systems that use memory-level flash devices expose new research challenges.
2. **Application-aware File System Design.** We start the file system research with the driving force of storage media. File system evolution or revolution can also be driven from application characteristics. Even more, characteristics of both applications and storage media can be matched. For instance, big data applications have different requirements on capacity, consistency, and accuracy. Characteristics of flash memory, such as error patterns, can be explored for relaxed accuracy in big data platforms.
3. **Distributed System Design on Flash Storage.** Opportunities in distribute systems that are brought by flash memory have not been well researched. Flash memory has characteristics, e.g, the no-overwrite property and the low I/O latency, which can change the designs in distributed protocols. Distributed system using flash storage is also a challenging but interesting research area.

7. Conclusion

Flash memory is not merely a fast storage media, but provides opportunities in new storage system designs. In this paper, we describe our work on different aspects of flash file system designs, including the hardware support, the storage management and the namespace management. In the hardware support aspect, our proposed LightTx protocol is designed in hardware to support storage consistency at low overhead, leveraging the no-overwrite property of flash memory with the knowledge of the birth and death of a transaction. In the storage management aspect, as far as we know, we are the first to identify the problem of *write amplification from file systems*. Our proposed OFSS removes duplicated functions that used to be in both file systems (software) and FTLs (hardware), and enables co-designs of software and hardware. By doing so, write traffic in file systems is significantly reduced, and the lifetime of flash storage is extended. In the namespace management aspect, our proposed ReconFS changes the write-back way of directory tree that has been used for forty years, based on the property of unbalanced read and write cost of flash storage. The new namespace management dramatically reduces the write overhead. With preliminary research on flash file systems, we believe that flash memory provides more opportunities than fast storage media and the I/O stack expects a revolution to exploit the potentials of flash memory.

References

[1] Btrfs. http://btrfs.wiki.kernel.org.

[2] Ext4. https://ext4.wiki.kernel.org/.

[3] FusionIO Virtual Storage Layer. http://www.fusionio.com/products/vsl.

[4] Trim. http://en.wikipedia.org/wiki/Trim (computing).

[5] Fusionio iodrive octal data sheet. http://www.fusionio.com/load/-media-/1shm1l/docsLibrary/FIO_DS_Octal.pdf, 2012.

[6] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D Davis, Mark S Manasse, and Rina Panigrahy. Design tradeoffs for SSD performance. In *Proceedings of 2008 USENIX Annual Technical Conference* (*USENIX ATC*), Berkeley, CA, 2008. USENIX.

[7] David G. Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, and Vijay Vasudevan. FAWN: A fast array of wimpy nodes. In *Proceedings of the 22nd ACM SIGOPS Symposium on Operating Systems Principles* (*SOSP*), pages 1-14, New York, NY, USA, 2009. ACM.

[8] Mahesh Balakrishnan, Dahlia Malkhi, Vijayan Prabhakaran, Ted Wobbler, Michael Wei, and John D. Davis. CORFU: A shared log design for flash clusters. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation* (*NSDI'12*), pages 1-14, Berkeley, CA, 2012. USENIX.

[9] Mahesh Balakrishnan, Dahlia Malkhi, Ted Wobber, Ming Wu, Vijayan Prabhakaran, Michael Wei, John D Davis, Sriram Rao, Tao Zou, and Aviad Zuck. Tango: Distributed data structures over a shared log. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles* (*SOSP*), pages 325-340. ACM, 2013.

[10] Randal E Bryant and David R O'Hallaron. *Computer Systems: A Programmer's Perspective*. Prentice-Hall, 2008.

[11] Feng Chen, David A Koufaty, and Xiaodong Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems* (*SIGMETRICS*), pages 181-192. ACM, 2009.

[12] Feng Chen, Rubao Lee, and Xiaodong Zhang. Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing. In *Proceedings of the 17th IEEE International Symposium on High Performance Computer Architecture* (*HPCA*), pages 266-277. IEEE, 2011.

[13] Joel Coburn, Trevor Bunker, Meir Schwarz, Rajesh Gupta, and Steven Swanson. From ARIES to MARS: Transaction support for next-generation, solid-state drives. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles* (*SOSP*), pages 197-212. ACM, 2013.

[14] Jim Gray, Paul McJones, Mike Blasgen, Bruce Lindsay, Raymond Lorie, Tom Price, Franco Putzolu, and Irving Traiger. The recovery manager of the system R database manager. *ACM Computing Surveys*, 1981.

[15] Laura M Grupp, John D Davis, and Steven Swanson. The bleak future of NAND flash memory. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies* (*FAST*), Berkeley, CA, 2012. USENIX.

[16] Yang Hu, Hong Jiang, Dan Feng, Lei Tian, Hao Luo, and Shuping Zhang. Performance impact and interplay of ssd parallelism through advanced commands, allocation strategy and data granularity. In *Proceedings of the International Conference on Supercomputing* (*ICS*), pages 96–107. ACM, 2011.

[17] William K Josephson, Lars A Bongo, David Flynn, and Kai Li. DFS: A file system for virtualized flash storage. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies* (*FAST*), Berkeley, CA, 2010. USENIX.

[18] Changman Lee, Dongho Sim, Jooyoung Hwang, and Sangyeun Cho. F2FS: A new file system for flash storage. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies* (*FAST 15*), Santa Clara, CA, February 2015. USENIX.

[19] Youyou Lu, Jiwu Shu, Jia Guo, Shuai Li, and Onur Mutlu. LightTx: A lightweight transactional design in flash-based SSDs to support flexible transactions. In *Proceedings of the IEEE 31st International Conference*

on Computer Design (*ICCD*), pages 115-122. IEEE, 2013.

[20] Youyou Lu, Jiwu Shu, Jia Guo, Shuai Li, and Onur Mutlu. High-performance and lightweight transaction support in flash-based SSDs. *to appear in IEEE Transactions on Computers.*

[21] Youyou Lu, Jiwu Shu, and Wei Wang. ReconFS: A reconstructable file system on flash storage. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies* (*FAST*), pages 75-88, Berkeley, CA, 2014. USENIX.

[22] Youyou Lu, Jiwu Shu, and Weimin 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* (*FAST*), Berkeley, CA, 2013. USENIX.

[23] Youyou Lu, Jiwu Shu, and Peng Zhu. TxCache: Transactional cache using byte-addressable non-volatile memories in SSDs. In *Proceedings of the 3rd IEEE Nonvolatile Memory Systems and Applications Symposium* (*NVMSA*). IEEE, 2014.

[24] Changwoo Min, Kangnyeon Kim, Hyunjin Cho, Sang-Won Lee, and Young Ik Eom. SFS: random write considered harmful in solid state drives. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies* (*FAST*), Berkeley, CA, 2012. USENIX.

[25] D. Nellans, M. Zappe, J. Axboe, and D. Flynn. ptrim ()+ exists (): Exposing new FTL primitives to applications. In *The 2nd Annual Non-Volatile Memory Workshop*, 2011.

[26] Xiangyong Ouyang, David Nellans, Robert Wipfel, David Flynn, and Dhabaleswar K Panda. Beyond block I/O: Rethinking traditional storage primitives. In *Proceedings of the 17th IEEE International Symposium on High Performance Computer Architecture* (*HPCA*), pages 301 – 311. IEEE, 2011.

[27] Jiaxin Ou, Jiwu Shu, Youyou Lu, Letian Yi, and Wei Wang. EDM: An endurance-aware data migration scheme for load balancing in SSD storage clusters. In *Proceedings of the 28th IEEE International Parallel and Distributed Processing Symposium* (*IPDPS*), pages 787-796, May 2014.

[28] Vijayan Prabhakaran, Thomas L. Rodeheffer, and Lidong Zhou. Transactional flash. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation* (*OSDI*), pages 147-160, Berkeley, CA, 2008. USENIX.

[29] Wei Wang, Youyou Lu, and Jiwu Shu. p-OFTL: an object-based semantic-aware parallel flash translation layer. In *Proceedings of the Conference on Design, Automation and Test in Europe* (*DATE*), page 157. European Design and Automation Association, 2014.

[30] Jisoo Yang, Dave B Minturn, and Frank Hady. When poll is better than interrupt. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies* (*FAST*), Berkeley, CA, 2012. USENIX.

[31] Yiying Zhang, Leo Prasath Arulraj, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. De-indirection for flash-based SSDs with nameless writes. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies* (*FAST*), Berkeley, CA, 2012. USENIX.

약 력

Youyou Lu

2015 Ph.D, Computer Science and Technology, Tsinghua University, P. R. China
2013 Visiting Scholar, Electrical and Computer Engineering, Carnegie Mellon University, USA
2009 Bachelor, Computer Science and Technology, Nanjing University, P. R. China

Jiacheng Zhang

2013~current Ph.D Student, Computer Science and Technology, Tsinghua University
2013 Bachelor, Software Engineering, Harbin Institute of Technology

Jiwu Shu

2014~current The Ministry of Education Chang Jiang Scholar Professor, Tsinghua University, P. R. China
2009 Winner of The National Science Fund for Distinguished Young Scholar, P. R. China
Jiwu Shu
2008~current Vice President, National Engineering Laboratory for Disaster Backup and Recovery, P. R. China
2008~current Academic Committee, State Key Laboratory of High-end Server and Storage Technology, P. R. China
2006~current Vice President, Technical Committee of Information Storage Technology, P. R. China Computer Federation (CCF TCIST)
2006~current Professor, Tsinghua University, P. R. China
2001~2006 Associate Professor, Tsinghua University, P. R. China
2000~2001 Assistant Professor, Tsinghua University, P. R. China
1998~2000 Postdoc, Computer Science and Technology, Tsinghua University, P. R. China
1998 Ph.D, Computer Science and Technology, Nanjing University, P. R. China