2013 IEEE 16th International Conference on Computational Science and Engineering

A New Exploration to Build Flash-based Storage Systems by Co-Designing File System and FTL

Wenwei Qiu, Xiang Chen, Nong Xiao, Fang Liu, Zhiguang Chen
State Key Laboratory of High Performance Computing, National University of Defense Technology
Changsha, China
{wenweiqiu, xiangchen, nongxiao, liufang, chenzhiguang}@nudt.edu.cn

***Abstract*-Nowadays, data storage and management has become an increasingly significant issue in the world of big data. With its density increasing and price declining, NAND flash has become a ubiquitous storage media in both enterprise and academic community. Flash chips are usually encapsulated into Solid State Drives (SSDs) by the Flash Translation Layer (FTL). SSDs exhibit the same interface as hard disk does thus are applicable to traditional file systems. A large number of technologies have been developed to improve the performance of SSD and SSD-oriented file systems. However, most prior works focused on either FTL or file system, but failed to combine them together. We argue that, it will gain more benefits if file system and FTL cooperate with each other. Contributions of our work include the following aspects. First, we introduce a new method to exploit flash memory by co-designing file system and FTL. Second, we implement out-of-place update at page granularity in file system by changing logical address allocation module to reduce the size of mapping table in FTL. Third, we provide file system with channel allocation by changing FTL and introduce three channel allocation schemes to improve SSDs performance. The evaluation results show that our co-design method gains similar performance as page mapping scheme with small mapping table. With more information to guide channel allocation, the average request response time is reduced by about 20% and throughput is improved by about 24% compared with traditional round-robin scheme. The metadata management in file system and data transmission overhead is negligible.**

Keywords-co-design; FTL; file system; parallel; mapping table

I. INTRODUCTION

The creation of digital data is occurring at a record rate and we are entering the age of Big Data [1]. The world of Big Data requires a high performance storage system to store and manage the big volume of data. Compared to traditional Hard Disk Drive (HDD), NAND flash memory based Solid State Drive (SSD) has the following advantages: higher bandwidth, more compact size, lower energy consumption and higher shock resistance. During the last two decades, the density of NAND flash memory has been increased constantly, e.g. Micron has released 1Tb MLC product [2], and the price per bit has fallen. As a result, the SSD is becoming a combination or substitution of HDD in high-end storage systems and a contributing technique in the world of Big Data.

Flash Translation Layer (FTL) performs address management, garbage collection, wear leveling and so on. Address management takes charge of logical and physical address transformation through different mapping schemes. Mapping scheme affects the size of mapping table and the performance of SSD. Many works endeavor to use less DRAM space to gain higher overall performance [3-6]. However, to the best of our knowledge, there are no works focus on co-designing file system and FTL, which makes sense in enhancing higher performance. We maintain a coarse-grained (block-level) mapping in FTL to reduce the memory consumption in SSD while maintain a fine-grained (page-level) mapping in file system to support fine-grained update. The Trim operation is employed to improve garbage collection.

SSD generally contains several channels. Traditionally, the FTL is responsible for allocating I/O requests to different channels. The FTL cannot capture semantics of the upper file system. Due to the limited information in FTL, the channel allocation scheme is often very simple [7]. Meanwhile, file system contains much more information than FTL, such as file name, logical address allocator, spatial locality and temporal locality and so on. We can utilize these hints to guide channel allocation to enhance the performance of SSD. We shift the executor of channel allocation from FTL to file system. The file system explicitly schedules I/O requests to different channels. Furthermore, we propose three channel allocation schemes to utilize the functionality. First, we allocate one file to all channels evenly to improve read performance. Second, we adjust read/write request to reduce operation interplay. Third, we adjust channel parallel depth according to I/O size to exploit other level parallelism.

In this paper, we propose a new methodology by co-designing file system and FTL to exploit the parallelism of

978-0-7695-5096-1/13 $31.00 © 2013 IEEE
DOI 10.1109/CSE.2013.138

IEEE computer society

flash memory without changing the interface connecting file system and SSD. Our contributions are as follows:

- We reduce the size of mapping table in FTL by co-designing file system and FTL. The average response time of co-design method is similar to page mapping scheme.
- We improve the channel parallelism performance by fully utilizing the information in file system. The effects of three channel allocation schemes are accumulative when we combine them.

We present a simulation-based study of our proposed co-design strategy. The evaluation results show that nearly 20% of average request response time reduction and nearly 24% of throughput improvement compared with page mapping FTL adopting round-robin scheme. The exploration to build flash-based storage systems will contribute to data store and data management in Big Data world.

The remainder of this paper is organized as follows: Section Ⅱ introduces the background and motivation of this paper. Section Ⅲ explains the detail of co-design between file system and FTL. Section Ⅳ shows the experimental results. Finally, we conclude this paper in section Ⅴ.

II. Background and Motivation

A. Flash Translation Layer

In a typical NAND flash memory [8], one device contains one or more chips, one chip contains one or two dies, one die contains two planes, one plane contains 2048 blocks, and one block contains 256 pages. There are three characteristics of flash memory. First, a page should be erased before being programmed. Second, the read and program operation are performed in the granularity of page, while erase operation is performed in the granularity of block. Third, each block has a limited erasure cycle before it is worn-out. As a result, file system cannot access flash memory directly in the same manner as accessing an HDD. A special layer called Flash Translation Layer (FTL) was proposed to hide these peculiarities so that file system can access flash memory. To alleviate write amplification brought by erase-before-write, out-of-place update is proposed to reduce erase and data movement overhead. When there is insufficient free space, garbage collection was triggered to reclaim invalid space. To lengthen the lifetime of SSD, wear leveling was proposed to ensure every block wears evenly. To enhance performance and lengthen the life time of SSD, a certain size of DRAM is used to cache hot data and reduce data written back to flash memory.

There are three mapping schemes for FTL. Page mapping has good performance for both read and program operations at the expense of large DRAM memory size. Block mapping, on the contrary, requires little DRAM memory space but goes with tremendous write amplification causing by update operation. Hybrid mapping is a tradeoff between DRAM memory usage and performance. Gupta et al. [6] proposed Demand-based Flash Translation Layer (DFTL), which is a page mapping scheme. Although DFTL saves DRAM space by storing its complete mapping table in flash, it incurs page mapping lookup overhead for workloads with less temporal locality.

B. Channel Parallel Scheme

In a typical SSD, NAND flash memory array constitutes several channels. Therefore, the SSD exhibits channel level parallelism. Kang et al. [9] investigated striping, interleaving, and pipelining optimization techniques to exploit the channel parallelism of SSD. Park et al. [10] designed an FTL for multi-channel/multi-way NAND flash-based storage devices (NFSD). Kang and Park's work focused on the design of FTL to explore parallelism, while our work combines file system with FTL to enhance parallelism.

Shin et al. [11] proposed six different static allocation schemes and five of them adopt round-robin scheme in channel level. Yang et al. [7] compared static allocation schemes with dynamic allocation schemes among varied workloads. They maintained that static allocation scheme outperforms dynamic allocation scheme in serving read requests. But our co-design evaluation results show that if file names are taken into account, the dynamic allocation outperforms static allocation in serving read requests.

Prior works have exploited the channel parallelism of SSD in FTL, but the allocation scheme is simple due to limited information in FTL. For one thing, the read or write requests issued by the file system may be arbitrary. For another, FTL can only recognize the logical address, size and types (read or write) of the request. Generally, one file may be distributed to different channels unevenly.

Fig. 1 shows an example of data distribution under round-robin manner. ByteOffset denotes the offset of request from start of disk in bytes; IOSize denotes the size of request in bytes. Assume that there are 8 channels in an SSD and the page size of flash memory is 4 KB. When the first write request of file A arrives, the FTL allocates channel 0 with 2 pages and other channels with one page. After the second and third write requests are issued, the distribution of file A is channel 0 contains 3 pages while channel 1 to 3 only contains 1 page. Because file A is distributed unevenly, the response time of read request to file A is determined by channel 0.

$$\text{Imbalance rate} = \frac{\text{max_channel} - \text{min_channel}}{\text{average number}} \quad (1)$$

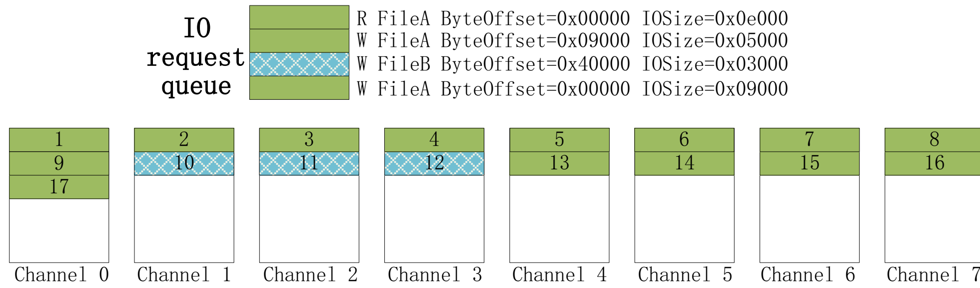

Fig. 1 An example of round-robin channel allocation scheme

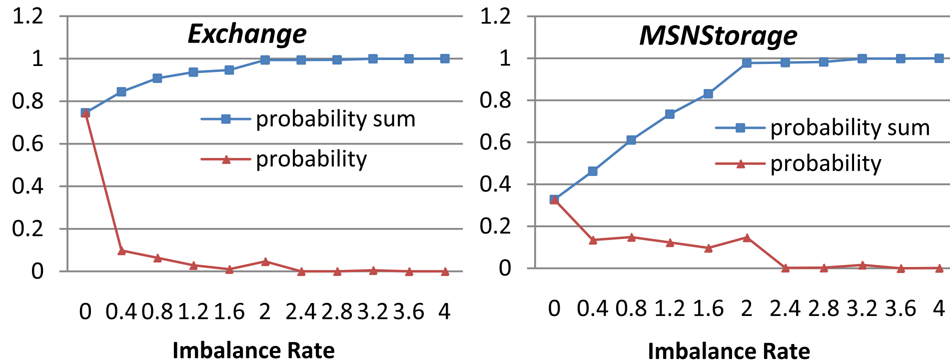

Fig. 2 Imbalance rateunder round-robin allocation scheme

We define ***Imbalance rate*** to characterize file distribution in (1). The ***max_channel*** and ***min_channel*** denote the maximum and minimum number of pages of one file among 8 channels respectively. The ***average number***, which is expressed in integer, denotes the largest number of pages among all channels if a file is evenly distributed. The imbalance rate is 0 if the file is less than 8 pages as well as no channel contains more than 2 pages. The larger the imbalance rate is, the more unevenly the file is distributed.

Fig. 2 shows the probability and cumulative distribution function of the imbalance rate under round-robin allocation scheme among different files in workloads *Exchange* and *MSNStorage* [12][13]. Nearly 26% of files are distributed unevenly and 6.3% of files' imbalance rate exceeds 1 in *Exchange*. Nearly 67% of files are distributed unevenly and 27.9% of files' imbalance rate exceeds 1 in *MSNStorage*. It is a common phenomenon that a file is distributed unevenly under round-robin allocation scheme.

C. Potential Benefit from File System

File system connects operating system and storage media. It organizes files and directories, and manages the address space of storage devices. File system contains file name, directory, file length, creation time, device type, user ID, group ID, and other attributes of file. Prior works have explored flash memory by FTL [14], but few works made use of file system to reduce the size of mapping table and improve SSD channel parallelism.

We take Ext3 file system as an example. The i-node is used to store the metadata of a file or directory. There are 12 direct block pointers and several indirect block pointers in i-node. The direct block pointers store file logical address pointers. The file logical address can be as small as page level. We can utilize the direct and indirect block pointers to store page logical address. In that case, the mapping scheme of FTL is changed from page mapping to block mapping so as to reduce the size of mapping table.

File system manages file name and file length while FTL does not have these information. By changing the logical address allocation scheme in file system and making full use of some functionality provided by FTL, the channel parallelism of SSD can be exploited more thoroughly.

JFFS2 [15], UBIFS [16], and YAFFS [17] are widely used flash based log-structured file systems. These file systems have taken the characteristics of flash memory into consideration to facilitate file operations. But these file systems aim at

embedded applications rather than high performance applications.

Lu et al. [18] proposed object-based flash translation layer design (OFTL) to reduce write amplification from file system so as to extend lifetime of flash memory. Qiu et al. [19] proposed a hybrid file system NVMFS to improve random write in NAND-flash SSD. NVMFS distributes metadata and hot file data on NVRAM while storing other file data on SSD to make full use of NVRAM and SSD. OFTL and NVMFS are similar to our work by co-designing file system and FTL, but our work focuses on mapping policy and channel allocation scheme.

III. Design and Implementation of Co-design

In this section, we describe the details of our co-design between file system and FTL. First, we describe our block mapping scheme in FTL with the help of file system. Then we describe the functionality of channel allocation in file system provided by FTL. Finally, we describe three channel allocation schemes to enhance performance.

A. Co-design to Reduce Mapping Table

To reduce the size of the mapping table, we adopt block mapping scheme in FTL. For a NAND flash memory which contains 256 pages in a block, the size of mapping table in block level is reduced by 255/256 compared with page mapping. However, block-level mapping usually supplies poor performance. This work combines file system and FTL to enhance the performance in a block mapping based FTL.

We change the logical address allocation module to implement out-of-place update in file system so as to improve SSD performance. We propose two tier logical block addresses in file system. First Tier Logical Page Address (FTLPA) denotes the attribute of the logical page address (LPA), which has valid, invalid or free status (shows in Fig. 3). File system allocates logical address according to the logical address status in FTLPA. Second Tier Logical Page Address (STLPA) stores in the direct or indirect block pointers of file i-node. Both two tier logical block addresses track logical address in page level. Because FTLPA just describes the attribute of the LPA sequence, the size of FTLPA is very small. The modification of file system is modularize and easy to migrate to other file systems.

We implement out-of-place update in the following four steps. For an overwrite operation, file system first checks the old LPA by searching STLPA. Second, file system invalidates the old LPA by setting its status to invalid in the FTLPA. Third, file system allocates a new LPA from allocating logical block. At the same time, the attribute of the new LPA in FTLPA is

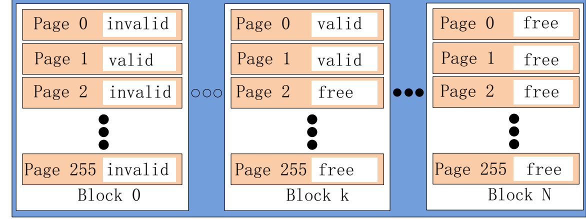

Fig. 3 First Tier Logical Page Address (FTLPA)

changed from free to valid status. Finally, file system updates STLPA in i-node and sends request to FTL.

Traditional file system allocates several consecutive logical addresses when a write request arrives. In our co-design file system, it allocates several logical addresses in page level rather than consecutive logical address. As a result, every page of data corresponds with one command, which will cause command transmission overhead. But the size of one page data is several orders of magnitude larger than a command, the transmission overhead between file system and SSD is negligible.

As out-of-place update is performed by file system, it needs a scheme to reclaim LPA. ATA interface proposed the TRIM command to send file delete notifications to SSD [20]. File system will incur some pages of data that are useless. The TRIM command can reduce the no-in-place-write overhead caused by subsequent overwrites. We take advantage of TRIM command to reclaim the LPA and further exploit it to help garbage collection.

Fig. 3 shows an example of out-of-place update and garbage collection. The page 1 of block 0 is valid and the other page of block 0 is invalid. Block *k* is the updating block and the updating page is 2.

1) Out-of-place Update. Assume that an overwrite operation happens in page 1 of block 0. File system will not sent an overwrite operation directly. Instead, it will allocate a new page (block k, page 2) in updating block to the request and set the status of old page (block 0, page 1) to invalid status. At the same time, file system changes the i-node of the file to the new logical address.

2) Garbage Collection. Assume that the free space of the whole storage system is under a given threshold, file system will trigger garbage collection to reclaim invalid LPA. File system first checks the FTLPA, and then chooses those logical blocks containing few valid data as victim. If block 0 is chosen as a victim, the status of page 1 is valid and its data need to be moved to another block. Then file system will send a block TRIM operation to SSD. The block TRIM operation will trigger an erase operation in SSD, which will contribute to garbage collection in FTL. The threshold is set as 5% of the whole SSD capacity. When the garbage collection reclaims

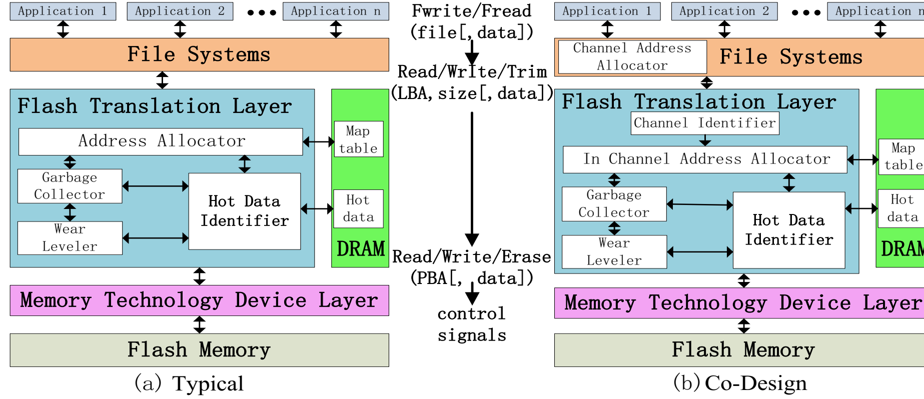

Fig. 4 Architecture of NAND flash memory-based storage systems

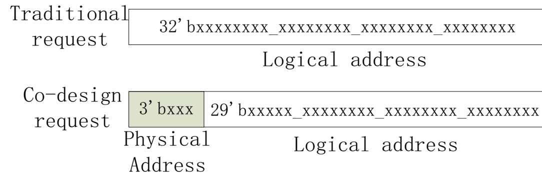

Fig. 5 Request structure

more than 10% of the whole SSD capacity, it will stop until next trigger operation.

B. Co-design to Improve Channel Parallelism

Fig. 4(a) shows a typical architecture of NAND flash memory-based storage systems. FTL plays the role of address allocator, garbage collector, wear leveler, hot data identifier and so on. DRAM keeps mapping table and hot data. Because DRAM is a volatile media, the data store in DRAM should be written back flash memory before shutting down electricity and be read from flash memory at the startup time. Memory Technology Device provides three basic flash operations (read/write/erase) for FTL through managing flash memory.

Fig. 4(b) depicts the alteration of file system and FTL compared with traditional NAND flash memory-based storage systems. Because file system contains more information than FTL, it will gain more benefit if file system plays the role of channel allocation. File system adds a Channel Address Allocator to distribute request to 8 channels. FTL adds a Channel Identifier to identify LBA sent by file system. At the same time, the Address Allocator is changed to In Channel Address Allocator.

Fig. 5 shows the request structure of an 8 channels SSD. We assume that traditional file systems use 32 bits to denote the logical address. In our co-design file system, we use the high three-bit to denote physical address. The three-bit physical address determines which channel the request belongs to. File system can send a request to one designate channel of the SSD by setting the three bit physical address. When FTL receives a write request, the Channel Identifier first interprets the request which channel it belongs to. Then FTL allocates a physical address of the channel it belongs to by In Channel Address Allocator.

To assist the functionality that file system take over channel allocation from FTL, the garbage collector should assure that every channel has similar free space and wear leveler should assure that every channel has similar worn-out level. Only in that case, can the storage system achieve overall good performance and endurance.

C. Channel Allocation Scheme

In this section, we describe three channel allocation schemes to make use of the functionality of channel allocation in file system to enhance the performance of storage system. We propose these three channel allocation schemes under different conditions, and the effect will accumulate when we combine them.

1) File Channel Fairness Scheme: To improve the read performance, Channel Address Allocator distributes every file to 8 channels evenly. We name this channel allocation scheme as File Channel Fairness Scheme. As the probability of one file being read is very high, the response time of read request is determined by the channel which contains the largest size of the file. In our co-design storage system, file system has the information of file name and the prerogative of channel allocation. Therefore, we can make full use of them in serving read request. When allocating new logical address to a write request, file system first calculates the size of the request. If the request size is k*8 (k=1, 2, 3…) times of the page, the request

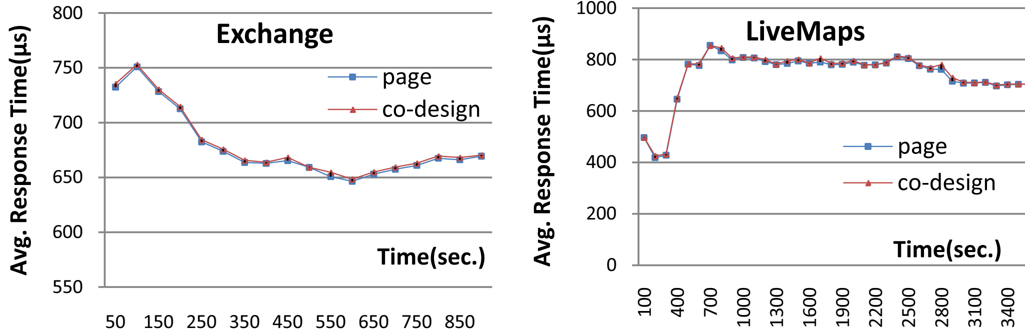

Fig. 6 Average response time

will be served by round-robin manner. Otherwise, file system will check the STLPA of the file in the i-node and calculates the file distribution among 8 channels. Then file system selects logical address from the channel that contains the least pages of the file.

The File Channel Fairness Scheme may disturb the traditional round-robin allocation scheme and degrade the performance of the SSD. To solve this problem, we adopt a counter for each channel to track the number of pages issued to the channel. When a file is already fairly distributed among 8 channels, we take the counter into account. File system will select those channels that are not so busy to issue the request. Therefore, every channel will be busy with fairly request number.

2) Read Busy Scheme: To reduce the read/write interplay with each other, Channel Address Allocator allocates less write requests to the channel which is occupied with read requests. We name this channel allocation scheme as Read Busy Scheme. Flash memory has a buffer to store data that written into flash or read out from flash. When two requests are both read or write operations, the two requests can utilize the buffer to pipeline the operation so as to improve throughput. Due to the buffer have different effects on read and write operations, different types of operation can only be issued in sequence. The read operation issued by user is arbitrarily, so we can schedule the write operation to mitigate the operation interplay. When the write request comes, we allocate the logical address in which channel is not busy with read operation. The Read Busy Scheme can reduce the response time of the request as well as improve the throughput of the SSD.

3) Dynamic Depth Scheme: To exploit other internal parallelism of SSD and reduce the DRAM permission exchange overhead, Channel Address Allocator adjusts the channel parallel depth according to the request length. We name this channel allocation scheme as Dynamic Depth Scheme. There are three levels parallelism except channel parallelism in SSD: chip level, die level and plane level. It will gain more benefits if we make full use of the other level parallelisms.

If the request size is larger than a threshold, Channel Address Allocator sends several sub requests to one channel at the same time. FTL can further use interleave technique and pipeline technique to exploit chip level or die level parallelism. Meanwhile, FTL can utilize Plane Command (plane read, plane write, and plane erase) to exploit plane level parallelism. We set the threshold as 16 pages because we can optimize the parallel depth to more than one.

Because DRAM is accessed in a sole way, every module in FTL interacting with DRAM should get the permission. However, the transformation between different modules takes a certain period of time. FTL has several modules communicating with the DRAM. As a result, it will cause permission exchange overhead between different FTL modules. If increasing the depth of the request, we can mitigate this overhead.

File Channel Fairness Scheme can improve read performance in the workload that file distributed unevenly under round-robin scheme. Read Busy Scheme can reduce request response time in the workload that read requests are asymmetry among 8 channels. Dynamic Depth Scheme can improve the performance in the workload that large requests are dominant.

IV. EVALUATION

We simulate our co-design in a flash-based Solid State Drive (SSD) which contains 8 channels. The characteristics of the evaluated NAND flash memory are shown in TABLE I. We evaluate our co-design simulation on the following four traces: (1) *LiveMaps* (2) *Exchange* (3) *Development* (4) *MSNStorage* [12][13]. The read ratios of four workloads are 64.4%, 32.1%, 95.9% and 64.6%. The average IO request sizes of four workloads are 49.1KB, 20.2KB, 24.2KB and 9.99KB.

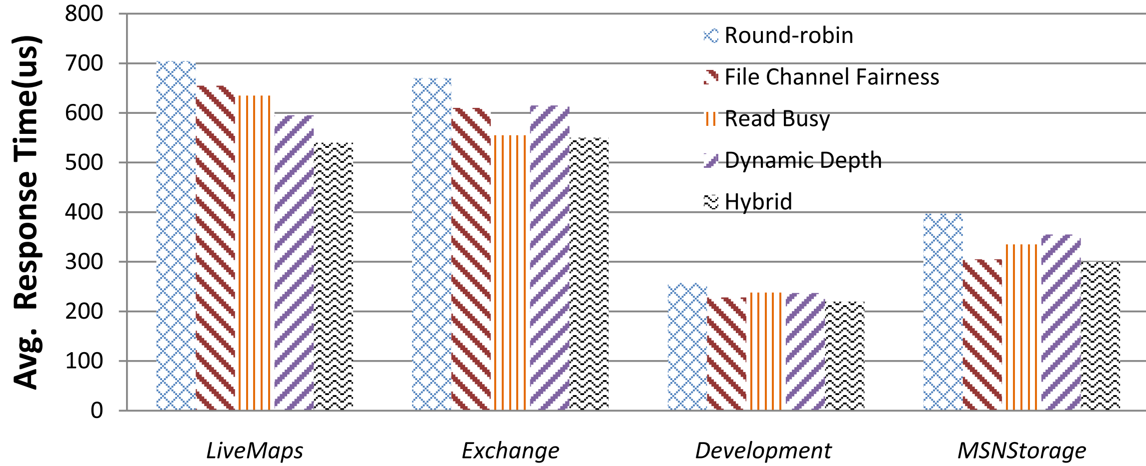

Fig. 7 Average Response Time

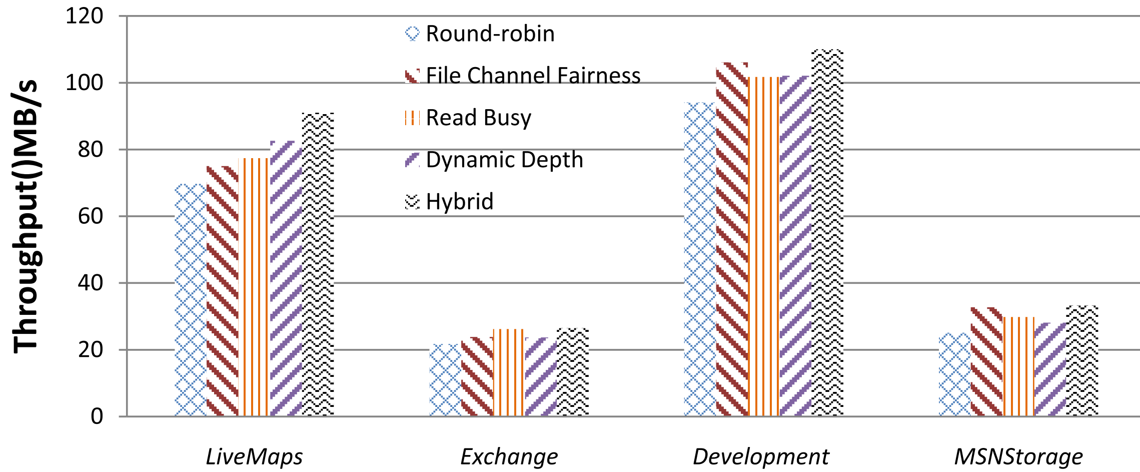

Fig. 8 Throughput

TABLE I THE CHARACTERISTIC OF EVALUATED MLC FLASH-MEMORY

Serial Access	25ns
Read	60µs
Write/Program	800µs
Erase	1.5ms
Page size	4KB
Block size	256pages

Fig. 6 compares the average response time of I/O requests between page mapping and our proposed co-design scheme. Although our co-design scheme adopts block mapping in FTL, the average response time is similar to that of page mapping scheme. The reason is that file system allocates the logical address in page level and implements out-of-place update. The sub request that file system sends to FTL will not trigger write amplification.

We evaluate the average response time and throughput of three channel allocation schemes in four workloads. And then we combine three channel allocation schemes together to build hybrid allocation scheme. Fig. 7 and Fig. 8 show the comparison of average response time and throughput achieved by round-robin allocation scheme, File Channel Fairness allocation scheme, Read Busy Scheme, Dynamic Depth scheme, and hybrid allocation scheme separately. In our co-design File Channel Fairness allocation scheme, the file is allocated to 8 channels evenly. The time consume in meta-data management is negligible. The request average response time

of File Channel Fairness allocation scheme is reduced by 7.0% to 23.1%. The throughput of File Channel Fairness allocation scheme is increased by 7.5% to 30.1%. Although the *Development* trace is read dominant (read 95.9% write 4.1%), the benefit gain from File Channel Fairness allocation scheme is little. Because the imbalance rate of *Development* under round-robin scheme is small. On the contrary, the *MSNStorage* gains much benefit due to large imbalance rate under round-robin scheme. The average response times of Read Busy Scheme is reduced by 7.4% to 17.2%. The throughput of Read Busy Scheme is increased by 8.0% to 20.7%. Because *Exchange* is read asymmetry among 8 channels, the benefit gain from Read Busy Scheme is effective. The average response time of Dynamic Depth scheme is reduced by 7.8% to 15.5%. The throughput of Dynamic Depth scheme is increased by 8.4% to 18.4%. Because *LiveMaps* contains many large requests, the benefit gain from Dynamic Depth scheme is more than other traces. The request average response times hybrid allocation scheme is reduced by 14.4% to 24.4%. The throughput hybrid allocation scheme is increased by 16.8% to 32.3%. The results show that hybrid channel allocation scheme is the most effective. The benefit of three channel allocation schemes in hybrid is accumulative.

V. CONCLUSION

In this paper, we propose co-design between FTL and file system to improve the performance of flash-based storage system. FTL adopts block mapping scheme. File system changes its logical address allocator to implement out-of-place update. File system utilizes channel allocation provided by FTL through three channel allocation schemes. The evaluation results show that although our proposed co-design method spends less DRAM to store mapping table, it can gain similar performance compared with page mapping. With more information to guide channel allocation, the performance is enhanced by about 24%. Our future work will explore hot data identification in file system to help FTL design. Building high performance flash-based storage systems will play significant role in the Big Data World.

ACKNOWLEDGMENT

We are grateful to the anonymous reviewers for their valuable suggestions to improve this paper. This work is supported by the National High Technology Research and Development 863 Program of China under Grant No. 2013AA013201, the National Natural Science Foundation of China under Grant Nos. 61025009, 61232003, 61120106005, 61170288, 61070198.

REFERENCES

[1] R. L. Villars, C. W. Olofson, M. Eastwood, “Big Data: What It Is and Why You Should Care,” White Paper, IDC, 2011

[2] MT29F1T08CUCABH8-6, http://www.micron.com/products/ nand-flash /

[3] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, “A space-efficient flash translation layer for compact flash systems,” IEEE Transactions on Consumer Electronics, 2002. Vol.48:366-375.

[4] S. W. Lee, et al, “A Log Buffer-Based Flash Translation Layer Using Fully-Associative Sector Translation,” ACM Transactions on Embedded Computing Systems, 2007. Vol. 6: No. 3, Article 18.

[5] D. Park, B. Debnath, and D. Du, “CFTL: a convertible flash translation layer adaptive to data access patterns,” In ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 10).

[6] A. Gupta, Y. Kim, and B. Urgaonkar, “DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings,” Architectural Support for Programming Languages and Operating Systems (ASPLOS 09).

[7] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang, “Performance Impact and Interplay of SSD Parallelism through Advanced Commands, Allocation Strategy and Data Granularity,” 25rd International Conference on Supercomputing (ICS 11).

[8] Datasheet, Micron 32Gb, 64Gb, 128Gb, 256Gb Asynchronous/Synchronous NAND Features.

[9] J. U. Kang, J. S. Kim, C. Park, H. Park, and J. Lee, “A multi-channel architecture for high-performance NAND flash-based storage system,” Journal of Systems Architecture. 2007. 53:644-658.

[10] S. Park, S. Ha, K. Bang and E. Chuang, “Design and analysis offlash translation layers for multi-channel NAND flash basedstorage devices,” IEEE Transaction on Consumer Electronics. 2009. Vol.55.

[11] J. Y. Shin, et al, “FTL design exploration in reconfigurable high-performance SSD for server applications,” 23rd International Conference on Supercomputing (ICS 09).

[12] http://iotta.snia.org/traces.

[13] D. Narayanan, A. Donnelly and A. Rowstron, “Write Off-Loading: Practical Power Management for Enterprise Storage,” Proc. of 6th USENIX Conference on File and Storage Technologies (FAST 08), pp. 253-267.

[14] E. Gal and S. Toledo, “Algorithms and data structures for flash memories,” In ACM Computing Survey’05, 2005, Vol.37:138-163.

[15] David Woodhouse. Jffs2: The journalling flash file system, version 2. http://sourceware.org/jffs2.

[16] Ubifs - ubi file-system. http://www.linux-mtd.infradead.org/doc/ ubifs.html.

[17] Yaffs. http://www.yaffs.net.

[18] Y. Lu, J. Shu, and W. Zheng, “Extending the lifetime of flash-based storage through reducing write amplification from file systems,” Proc. of 11th USENIX Coference on File and Storage Technologies(FAST 13), pp. 257-270.

[19] S. Qiu and A. L. N. Reddy, “NVMFS: A Hybrid File System for Improving Random Writein NAND-flash SSD,” 29th Symposium on Mass Storage Systems and Technologies (MSST 13).

[20] Frank Shu. Notification of Deleted Data Proposalfor ATA8-ACS2. http://t13.org/Documents/UploadedDocuments/docs2007/e07154r0-Notification_for_Deleted_Data_Proposal_for_ATA-ACS2.doc, 2007.