SCIENCE CHINA
Information Sciences

• RESEARCH PAPERS •

Special Focus

June 2011 Vol. 54 No. 6: 1129–1141
doi: 10.1007/s11432-011-4266-z

P3Stor: A parallel, durable flash-based SSD for enterprise-scale storage systems

XIAO Nong*, CHEN ZhiGuang, LIU Fang, LAI MingChe & AN LongFei

Department of Computer Science, National University of Defense Technology, Changsha 410073*, China*

Received September 11, 2010; accepted April 26, 2011

Abstract Driven by data-intensive applications, flash-based solid state drives (SSDs) have become increasingly popular in enterprise-scale storage systems. Flash memory exhibits inherent parallelism. However, existing solid state drives have not fully exploited this superiority. We propose P3Stor, a parallel solid state storage architecture that makes full use of flash memory by utilizing module- and bus-level parallelisms to increase average bandwidth and employing chip-level interleaving to hide I/O latency. To improve the bandwidth utilization of traditional interface protocols (e.g., SATA), P3Stor adopts PCI-E interface to support concurrent transactions. Based on the proposed parallel architecture, we design a lazy flash translation layer (LazyFTL) to manage the address space. The proposed LazyFTL adopts flexible super page-level mapping scheme to support multi-level parallelisms. It is able to distinguish hot data from cold data, and hot data identification enables LazyFTL to direct hot and cold data to separate physical blocks, which reduces page migrations when reclaiming blocks. As garbage collector migrates fewer valid pages, write amplification is significantly reduced, which in turn helps to extend the life span. Moreover, LazyFTL rarely triggers wear-leveling process. The lazy wear-leveling mechanism protects users' requests from being disrupted by background operations. With the guidance of hot data identification, an intelligent write buffer is used to reduce program operations to flash chips. This is meaningful in extending P3Stor's life span. The performance evaluation using trace-driven simulations and theoretical analysis shows that P3Stor achieves high performance and its life span is more than doubled.

Keywords flash memory, SSD, P3Stor, FTL, storage

Citation Xiao N, Chen Z G, Liu F, et al. P3Stor: A parallel, durable flash-based SSD for enterprise-scale storage systems. Sci China Inf Sci, 2011, 54: 1129–1141, doi: 10.1007/s11432-011-4266-z

1 Introduction

As the reliance on data-intensive applications continuously increases in various domains, disk-based storage devices are failing to meet the demands owing to high latency and low throughput. In contrast, flash memory outperforms hard disks in terms of its lower latency, lower power consumption, lighter weight, and shock resistance. Because of the increase in capacity and decrease in price, flash memory has attracted increasing attention in recent years. Compared with traditional hard disks, flash memory has several peculiarities making it unsuitable for direct application to current storage systems. First, a single flash memory chip is inadequate for large-scale storage systems in terms of both capacity and

*Corresponding author (email: nongxiao@nudt.edu.cn) performance. As such, it is better to explore a new architecture to aggregate sufficient flash chips. Second, flash chips do not support the specific interface protocols for hard disks. To be compatible with the software stack accumulated by traditional hard disks, flash chips need to be packaged as a solid state drive (SSD), which presents a hard disk-like interface. The software layer that hides the peculiarities of flash chips is called flash translation layer (FTL). SSD's performance is highly dependent on two aspects described above, the architecture and FTL design.

In this paper, we design a high performance parallel solid state storage device called P3Stor. P3Stor aims at high bandwidth, low latency, and long life span. To achieve these goals, we optimize both the architecture configuration and FTL design. Techniques employed in P3Stor are listed below.

• High-speed interface such as PCI-E is used to support multiple concurrent transactions. This improves the poor bandwidth utilization of traditional interface protocols.

• Module- and bus-level parallelisms are used to increase average bandwidth. In addition, chip-level interleaving is used to reduce I/O latency.

• P3Stor takes advantage of pipelining to accelerate the command interpretation process.

• To alleviate the negative impact of background operations, we adopt a flexible mapping scheme based on super pages. The fine-grained mapping table is able to distinguish hot data from cold data, which enhances the efficiency of FTL remarkably.

• The write buffer guided by hot data-aware mapping table not only responds to write requests rapidly, but also remarkably decreases write operations to flash chips thereby lengthening the life span of P3Stor.

• The lazy garbage collection and wear-leveling significantly reduce the overhead of FTL. They can efficiently prevent read requests from being congested by background operations.

The rest of this paper is organized as follows. Section 2 presents the background and related work. Section 3 describes principles of our work. Section 4 explains the architecture of P3Stor, while section 5 discusses the design of LazyFTL. Section 6 explores the design space of FTL and presents our performance evaluation. The final section concludes our work.

2 Background and related work

2.1 NAND flash memory overview

Compared with traditional hard disks, flash memory has several novel characteristics. A NAND flash chip is organized into several planes. A plane contains thousands of blocks, while a block consists of 64 or 128 pages [1]. A page is appended by a small spare Out-Of-Band area (OOB). Metadata, such as ECC (error correction code) and page states (free/valid/invalid), which are used to describe the page of data, can be written to the additional area.

Essentially, pages of flash chips can be accessed in parallel and at random. A typical flash-based device is made up of a package of flash chips, which are connected by multiple buses. Besides the external-chip parallelism among the buses, a flash chip with multiple planes also presents internal-chip parallelism among the planes. The external- and internal-chip parallelisms make it feasible for a flash-based device to achieve a higher aggregate bandwidth.

Flash memory has three operations: read, program, and erase. Program and erase are destructive operations. Too many program/erase operations on a block lead to data corruption. The nominal life span of a block in SLC NAND flash is 10^5 erasure cycles. For MLC flash, it is only 10^4 erasure cycles. When manufacturers pronounce the nominal life span of a chip, they guarantee that the bit error rate will be lower than 10^{-15} to 10^{-14}. However, Desnoyers et al. [2] pointed out that the measured life span of blocks varies greatly, and is often as much as 100 times higher than the manufacturer specifications.

2.2 Typical architectures based on flash memory

With the novel characteristics presented in the previous subsection, flash memory has attracted much attention over the past few years. Two typical architectures based on flash memory are briefly described below.

Gordon. Gordon [3] is flash-based system for massive parallel, data-centric computing. It leverages solid state drives, low-power processors, and data-centric programming paradigms to deliver enormous gains in performance and power efficiency. The basis is the adoption of solid state storage. Gordon exploits parallelism by two techniques. The first is aggressively pursuing dynamic parallelism between requests to flash array. The second is combining physical pages from several planes into super pages. There are three ways of creating super pages: horizontal striping, vertical striping, and 2-dimensional striping. Horizontal striping effectively creates a wider bandwidth bus, and increases the raw performance of the array for large reads. Vertical striping further enhances throughput by increasing bus utilization. Two-dimensional striping combines both horizontal and vertical striping schemes to generate an even larger super page. This scheme achieves greater bus bandwidth and lower I/O latency.

Hydra. Hydra [4] also exploits the parallelism among multiple flash chips to enhance storage performance. Techniques adopted to achieve this goal include the following: 1) Hydra adopts bus-level interleaving to generate a wider bandwidth by aggregating enough flash buses. 2) Chip-level interleaving is used to hide read latency. These two techniques are similar to the horizontal and vertical striping schemes adopted by Gordon. 3) Hydra uses a dedicated foreground unit with higher priority to handle read requests. This scheme prevents read requests from being delayed by background operations. 4) Hydra employs a write buffer to respond rapidly to write requests. The write buffer also makes it possible for programs to be performed in parallel. Hydra primarily focuses on parallelism. It emphasizes scheduling of multiple foreground and background units at system level. However, flash memory operations induced by interleaving are inflexible. Garbage collection and wear-leveling (explained in the next subsection) have not been optimized. Moreover, Hydra does not address reliability issue.

2.3 Introduction to FTLs

To hide the peculiarities of flash memory and emulate the interface of hard disk drives, flash chips are packaged into SSDs. The core of SSD is the FTL, which consists of three components: wear-leveling, garbage collection, and address mapping. Since each block has limited life span, it is important to distribute erase operations throughout the chip evenly to prevent some blocks from being corrupted too early. This is exactly what wear-leveling does. Since a page cannot be overwritten, an alternative method is to direct the updated data to another page and to mark the old page as invalid. FTL needs a garbage collection mechanism to reclaim these invalid pages. The Out-of-Place Update also leads to dynamic mapping between logical and physical addresses. A table, therefore, is necessary to maintain the mapping information. The state-of-art FTLs include BAST [5], FAST [6], SAST [7], KAST [8] and DFTL [9]. The focus of FTLs mentioned above is generally constrained to mapping schemes and hardly ever involves parallelism, wear-leveling or reliability. CAFTL [10] focuses on de-duplication at device level and can be seen as an add-on for the basic FTLs. CA-SSD [11] employs content addressable storage to exploit value locality, which helps to reduce the average response time.

3 Design principles

We try to build a high bandwidth, low latency, long life span SSD. The following principles guide our design.

Aggressively exploit parallelism. Typically, a page of a read request is subject to about a 20 μs read delay and a 100 μs bus delay for an SLC NAND flash chip. With a page size of 2 KB, the bandwidth for read requests is no more than 16.7 MB/s. A read request consisting of 64 consecutive pages must endure a latency of 7.68 ms. Neither bandwidth nor latency of a single chip with only a single plane is comparable to that of hard disks. Fortunately, external- and internal-chip parallelisms can overcome these disadvantages. Chip-level interleaving improves bus utilization significantly. A chip with four planes achieves a bandwidth of about 20 MB/s for both read and write. Bus-level interleaving produces an aggregate bandwidth, which can exceed the peak of host's interface protocol. As interleaving is the preliminary method to increase bandwidth, we adopt this method as well.

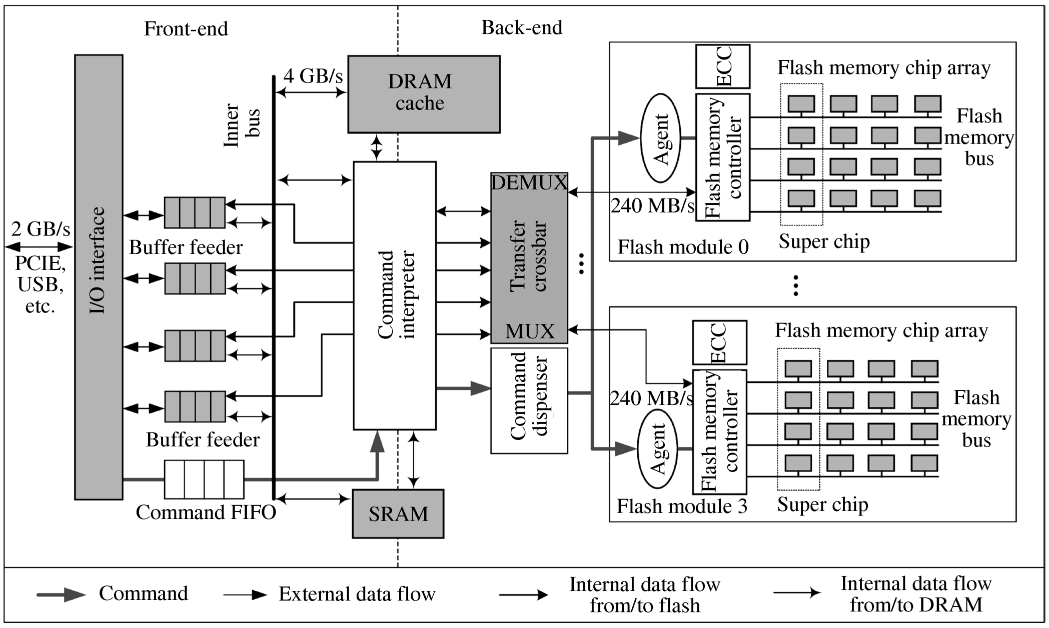

Figure 1 The overall architecture of P3Stor.

Write as few as possible. As each block has limited life span, it is important to reduce program operations to flash chips. As presented in the work of Soundararajan et al. [12], write requests from hosts exhibit great temporal and spatial localities. Most of the write requests are overwrites to a few data. A write buffer can significantly reduce the program operations to flash chips.

Do collect the garbage. When a block is recycled, if it has valid pages, garbage collector must move these valid pages first. Moving pages leads to extra read and write. The extra traffic not only impairs performance, but also wears out flash chips. If there are no valid pages in the block to be reclaimed, this negative impact disappears. So, we endeavor to reduce the number of valid pages in a block to be reclaimed.

Belittle wear-leveling. Wear-leveling distributes erasures over all blocks. This usually involves migrating pages in a block with the most erasing cycles to a block with the least erasing cycles. This process imposes even a greater negative impact on performance than garbage collection does. We argue that frequent wear-leveling is unnecessary, and that wear-leveling is not the only way to achieve higher reliability. This research employs a novel mechanism with little overhead, but which, nevertheless, guarantees reliability.

Endeavor to exhaust the lifecycle. Flash chips in the market have a nominal life span. However, Desnoyers et al. [2] pointed out that, even if a block has been erased more than the nominal number of times, it can still be of use. The measured lifetimes of blocks are usually as much as 100 times higher than the manufacturer specifications. This motivates us to make use of flash chips for as long as possible, while guaranteeing the expected reliability.

4 The P3Stor architecture

An overview of P3Stor is shown in Figure 1, with the main features of this architecture summarized below.

• To improve the poor bandwidth utilization of traditional interface protocols, P3Stor adopts high-speed interfaces, such as PCI-E, to support multiple concurrent transactions.

• To fully exploit the performance of the flash chip array, P3Stor adopts both module- and bus-level parallelisms to increase average bandwidth for multiple requests, and chip-level interleaving to reduce average latency.

• To accelerate the command interpretation process, we design a pipeline for command interpreter. The pipeline eliminates the bottleneck of centralized interpretations in traditional architectures.

4.1 Front-end and back-end

The overall architecture is divided into Front-end and Back-end as shown in Figure 1. They are described below.

Front-end. There are three major components in Front-end: command FIFO queue, buffer feeder, and DRAM for write buffer. When an I/O request arrives, the command FIFO queue accommodates it and immediately sends it to command interpreter. The command interpreter determines whether requested data locates in DRAM. For a read request, if the data hits in DRAM, it is immediately returned to corresponding buffer feeder, otherwise translation to a physical address is invoked. Similarly, when a write request arrives, data to be written is cached in one of the four buffer feeders. P3Stor is deployed with four buffer feeders and thus supports four simultaneous I/O requests from hosts. DRAM devotes to write requests. For a write request, once the data has been written to DRAM, the host is informed that the write request has been completed. However, for read requests missing from DRAM, the data fetched from flash chips is delivered to buffer feeder directly.

Back-end. The Back-end includes SRAM, command interpreter, command dispenser, agents, flash controllers, ECC encoders and decoders. SRAM is the cache for mapping table. P3Stor uses fine-grained mapping scheme. The mapping table is too large to be loaded into SRAM completely. We cache mapping information on demand. The command interpreter is responsible for translating logical addresses into physical addresses and for converting I/O requests to the format understood by flash controllers. The command dispenser dispatches commands to different agents according to physical address supplied by the command interpreter. The agents distribute different commands to flash controllers in out-of-order mode to exploit chip-level interleaving. Besides, the ECC encoder and decoder are used to detect and correct bit flips. Each flash memory controller is equipped with multiple encoders and decoders. As explained in section 5, P3Stor applies different ECCs to different data.

4.2 Multi-level parallelism

As illustrated in Figure 1, we exploit three different levels of parallelism. Details of them are given below.

Module-level parallelism. The entire flash array is divided into 4 modules. Each module is controlled by an agent. The command dispenser issues multiple commands to invoke different agents according to addresses of I/O requests. The agent is responsible for scheduling I/O commands to buses. In this way, all agents respond to I/O transactions concurrently. P3Stor dynamically balances the workload between the four agents.

Bus-level parallelism. In the P3Stor configuration, the data-paths of each module are 32 bits and can operate at 60 MHz in the FPGA prototype. In contrast, the datasheet of our selected flash memory indicates an 8-bit data path and 60 MHz operation frequency. To make up the bandwidth gap between the agent and flash chip, one reasonable configuration is to exploit the 4-way bus-level parallelism. Parallelism between the four buses achieves an effective bandwidth of 240 MB/s when using four 60 MB/s flash chips.

Chip-level parallelism. P3Stor supports chip-level interleaving of different chips sharing a bus, thereby hides the flash command latency. For example, after issuing an erase command to super-chip 0, the agent waiting for the erase command completion is not suspended. If other commands are inspected and found to be directed to different super-chips, they will be issued immediately to overlap the execution of erase. Afterwards, the agent examines the completion of all commands by issuing status check, and also issues arriving commands directed to free chips. In this way, chip-level parallelism can hide the I/O latency and increase bus utilization.

To support multi-level parallelisms, the address space is arranged as follows: the entire flash array is divided into 4 modules. Each module containing four buses is controlled by an agent as shown in

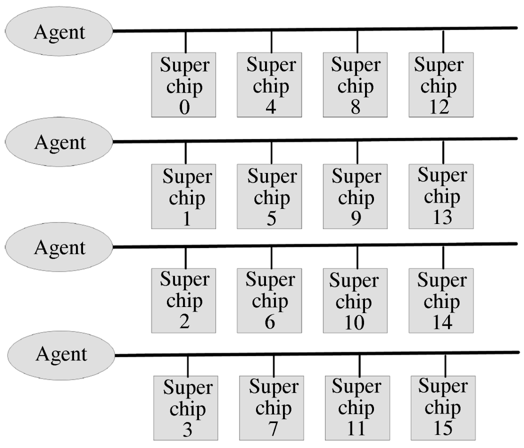

Figure 2 Address arrangement of modules.

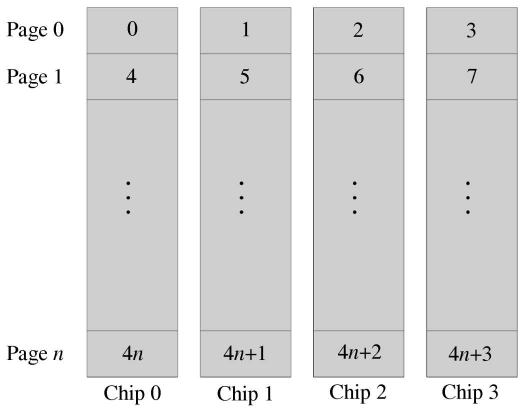

Figure 3 Address arrangement of a super chip.

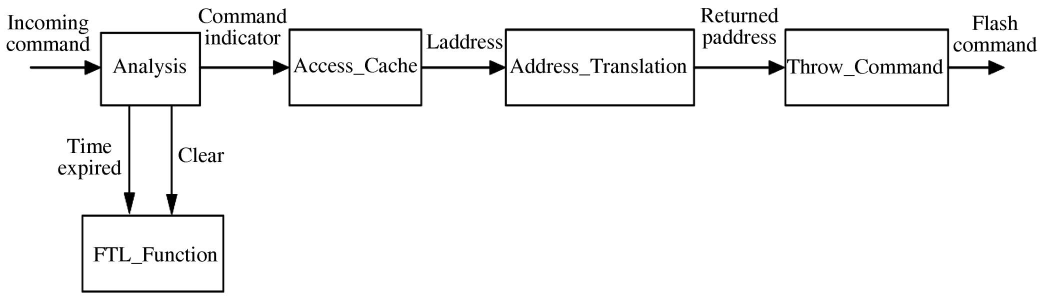

Figure 4 Pipeline of the command interpreter.

Figure 2. In each module, flash chips of the four buses are grouped into four super chips. Each super chip organizes four flash chips with the address arrangement as shown in Figure 3.

4.3 Command interpretation pipeline

To reduce the latency of each transaction, a multifunctional pipeline consisting of four stages is implemented in command interpreter (see Figure 4). The function of analysis stage is parsing I/O requests and converting them into several I/O commands at the granularity of super page. Besides that, the analysis stage is responsible for triggering background operations of FTL. For example, if no I/O request arrives for a long time, the analysis stage invokes garbage collection. The access cache stage checks whether requested data is cached in write buffer. If the data is missed, the address translation stage translates logical address into physical address with the help of page table cache. The physical address is further converted to a triple by the throw command stage. The triple designates target super chip, super block, and super page. The command is finally forwarded to corresponding agent.

5 The LazyFTL

Following the principles mentioned in section 3, we propose LazyFTL to complement the architecture described above. On the one hand, LazyFTL is lazy and resists reclaiming blocks that contain valid pages. It attempts to reduce programs by employing a write buffer and avoid wear-leveling. On the other hand, LazyFTL is smart. It adopts a super page-level mapping scheme for flexibility. It separates hot and cold data to improve the efficiency of garbage collection. It even enables P3Stor to survive beyond its nominal life span.

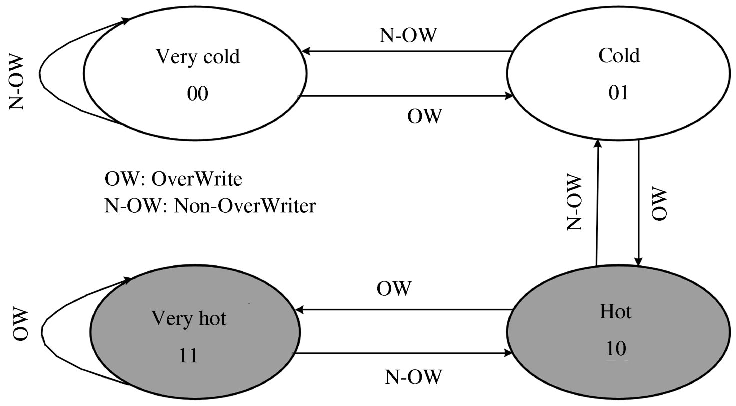

Figure 5 2-bit hot data predictor.

5.1 Hot page-aware page table management

For P3Stor, the chip-level interleaving packages four pages from different chips into a super page, which is the atomic access unit. LazyFTL adopts super page-level mapping scheme for flexibility. Management of the page table includes two aspects, the configuration of page table cache and hot page prediction.

The prototype of P3Stor is 256 GB. A super page contains four 2 KB physical pages. There are 32 M super pages. So, the mapping table contains 32 M entries. If each entry takes up 4 B, a physical page can accommodate 256 entries (the left 1024 bytes are reserved for future use). The whole page table consumes 256 MB. It is too costly to hold the entire mapping table in memory of SSD. Fortunately, localities exhibited by most workloads make it feasible to cache the recently used entries on demand. While the complete copy of mapping table is maintained on flash chips. Hereafter, the mapping information is referred as kernel data. The spatial overhead of kernel data is negligible. For a 256 GB SSD, there are only 256 MB kernel data. The spatial overhead is approximate 0.1% of the SSD capacity. The 256 MB kernel data holds 128 K physical pages. The data structure that indexes the 128 K pages is called page table directory. Each entry in the page table directory also takes up 4 B. Thus the page table directory consumes 0.5 MB. Page table directory is the root to boost the system. It is kept in a battery-backed RAM.

The cache is managed at the granularity of a page, which contains 256 entries. If some of the entries in a page are accessed, the whole page is fetched into cache. Entries in a page are also evicted out of the cache as a whole. To exploit temporal locality, the cache employs the 4-way set-associative structure. To exploit spatial locality, the cache employs a conservative prefetch policy. When a page of kernel data is demanded, the next page beyond the demanded one is also fetched into cache.

Flash is write-sensitive. We call the logical pages that are updated frequently, hot pages. The others are cold pages. It is useful to direct hot and cold pages to separate blocks. We call the process that distinguishes hot data from cold data, hot page prediction. Garbage collection benefits from hot page prediction. Since the page table cache is aware of all the access traces, LazyFTL performs hot page prediction in the page table cache.

LazyFTL adopts 2-bit predictors [13] to predict hot pages. Figure 5 reviews the principle of 2-bit predictor. All the super pages are divided into four groups, very hot, hot, cold, and very cold, which are represented by 11, 10, 01, and 00, respectively. If a very cold super page has been over-written many times, it will ultimately become a very hot super page. On the contrary, if a very hot super page has not been over-written for a long time, it will become a very cold one. Conversions between the states are illustrated in Figure 5. When a super page is accessed for the first time, LazyFTL fetches the mapping entry into cache. Exactly, LazyFTL fetches the whole page containing the entry into cache. If the super page has been overwritten, LazyFTL upgrades its hot degree. On the other hand, if the super page has not been overwritten until that its mapping entry is evicted out of cache, LazyFTL degrades its hot degree. As discussed above, a 2 KB physical page contains 256 mapping entries, which consume 1024 bytes. The remaining 1024 bytes are reserved for storing the states of predictors. LazyFTL maintains a

2-bit predictor for each entry. 256 predictors consume 64 bytes. When a page of entries is evicted out of cache, states of their predictors are also written to flash chips for future use.

5.2 Write buffer

As each block has limited life span, LazyFTL employs a DRAM as write buffer to reduce programs. The write buffer adopts 4-way set-associative mapping scheme. It supports two triggers to flush pages to chip arrays.

Capacity-threshold trigger: If the free capacity of write buffer is lower than a given threshold, flush operation is triggered.

Page table cache flush trigger: If the free capacity of page table cache is lower than a given threshold, the cache must flush some mapping entries to flash chips. If the flushed entry is dirty, corresponding user data in write buffer are flushed as well. As the page table cache is aware of hot and cold data, it always evicts cold entries out of cache. Corresponding cold user data is also flushed by write buffer. Guided by page table cache, the write buffer achieves high hit ratio and significantly reduces program operations.

When data is flushed, LazyFTL optimizes program operations in two ways. The first is performing programs in parallel by adopting bus- and chip-level interleaving. The other is directing hot and cold super pages to separate super blocks. Separating hot data from cold data enhances the efficiency of garbage collection.

5.3 Lazy garbage collection

Although interleaving increases the bandwidth, it imposes a negative impact on garbage collection. Suppose that a write request consists of 16 pages, the 4-way bus-level interleaving dispatches 16 pages to 4 buses, the 4-way chip-level interleaving dispatches 4 pages to 4 different chips. As a result, the 16 pages from a write request scatter across 16 physical blocks. Overwriting the 16 pages leaves each block with an invalid page. Interleaving causes invalid pages to be dispersed across more blocks.

LazyFTL overcomes this disadvantage in two ways. First, it adopts super page-level mapping scheme. The fine-grained mapping scheme enables all physical super pages to be able to accept updates. So, garbage collection will not be triggered until free blocks are used up. Usually, an SSD reserves some blocks that are prepared to replace worn-out blocks. The large redundant capacity ensures that garbage collection is rarely triggered. As garbage collection is delayed, more pages have their chances to become invalid, garbage collector will migrate fewer valid pages in recycling a block. Second, hot and cold pages are directed to separate super blocks. In addition, garbage collection is delayed for a long time by adopting super page-level mapping scheme. So, super blocks containing hot pages tend to have no valid pages. LazyFTL, therefore, almost moves no valid pages during garbage collection.

5.4 Lazy wear-leveling

LazyFLT hardly performs wear-leveling. Only when some super blocks' erasing cycles approach the nominal life span, LazyFTL moves some cold pages to them. Meanwhile, more powerful ECC, such as Reed-Solomon [14] code, is employed to guarantee reliability. Although Reed-Solomon code is more complex, as cold data is rarely accessed, the complex encoding and decoding process is rarely triggered. As hot data is stored in reliable super blocks, LazyFTL applies commonplace ECC to hot data for simplicity.

Desnoyers' experiments show that the measured life spans of blocks are often as much as 100 times higher than manufacturer specifications [2]. Even if the nominal life span has expired, the block can still be in service. LazyFTL makes full use of the actual life span by adopting a more powerful ECC, and thus extends the life span of P3Stor. Theoretically, LazyFTL can extend the actual life span as long as the powerful ECC can guarantee reliability. But, as bit errors happen more frequently, repeated encoding and decoding will impair performance. So, LazyFTL only stores cold data in nominally worn out blocks.

Hot data is stored in reliable blocks. When all blocks have exceeded their nominal life spans, there are no reliable blocks to store hot data. The SSD is ultimately worn out.

6 Design space exploration

We evaluate P3Stor by trace driven simulations. Subsection 6.1 introduces experimental setup. Subsection 6.2 presents design of page table. Subsection 6.3 discusses the write buffer. Comparison of garbage collection with and without hot data prediction is presented in subsection 6.4. The last subsection analyzes the life span.

6.1 Experimental setup

The experiments are based on trace driven simulations. PC1 and PC2 were collected from desktops running Windows XP. The applications include ftp, email, VMware, databases, and text editor tools [15–17]. Server1 and Server2 were collected from servers supplying a platform for online sales. Financial1 and Financial2 are traces from an OLTP application running at a financial institution [18] and made available by Storage Performance Council (SPC). Web1 and Web2 are read-dominant Web Search engine traces [19].

We used the well-known simulator, FlashSim [20]. FlashSim imitates all components of flash chip, including pages, blocks, planes, dies, packages, channels and buses. Developers of FlashSim had validated the performance thereof against a number of commercial SSDs for behavioral similarity. All simulations are based on the Samsung K9WAG08U1A chip [1]. A chip contains 4 planes, with each plane consisting of 2048 blocks. A block contains 64 pages. A page is 2 KB with a 64 B OOB area. Generally, SSDs contain redundant blocks. In our experiments, we assume that the extra capacity is 10% of the nominal capacity of SSD.

6.2 Page table management

In this subsection, we explore the management of page table cache. We believe that simplicity is as important as performance in designing a system. The primary cache replacement policy is 4-way set-associative scheme without prefetch. Within each set, we adopt LRU [21] for simplicity. Table 1 presents the hit ratios achieved by the 4-way set-associative policy and fully-associative mechanism in a 4 MB cache, respectively. As the table shows, the 4-way set-associative policy is comparable with the fully-associative one. In the page table, a page contains 256 entries indexing 256 super pages. Access to any super pages results in an access to the page of kernel data. Thus, spatial locality of user data requests is converted into temporal locality for kernel data requests. As temporal locality is consolidated, a simple replacement policy is sufficient for the page table cache. This is why 4-way set-associative policy is comparable with a fully-associative one. The 4-way set-associative policy is adopted due to its simplicity. To improve hit-ratio further, we integrate a conservative prefetch mechanism into the cache replacement policy. Whenever a page is demanded, we prefetch the consecutive page of kernel data. Hereafter, the 4-way set-associative replacement policy with a conservative prefetch mechanism is adopted for the page table cache.

There are two kinds of overhead for the page table, capacity overhead and I/O overhead. Kernel data consumes 0.1% of the capacity of P3Stor, which is negligible. I/O overhead is induced by requests for kernel data. It consists of two types. One involves fetching demanded entries into cache, while the other involves evicting dirty entries out. Table 2 analyzes the I/O overhead. The column, Request pages, denotes the total pages requested by hosts. Extra read fetches demanded entries into cache. Extra write evicts dirty entries out. As the last column shows, the total number of requested pages of kernel data is less than 5% of the total requested pages of user data. Specifically, extra writes are no more than 0.5% of user data writes. Write requests of kernel data hardly reduce the life span of P3Stor.

Table 1 Comparison of two types of cache configurations

Traces	Set-assoc (%)	Fully-assoc (%)
PC1	80.55	80.96
PC2	81.82	82.06
Server1	95.41	95.82
Server2	95.00	95.46
Financ1	99.40	99.43
Financ2	97.12	97.68
Web1	48.67	48.68
Web2	48.60	48.59

Table 2 I/O overhead of kernel data

Traces	Requested pages	Extra read (%)	Extra write (%)	Total (%)
PC1	140909866	0.804	0.279	1.083
PC2	184729385	0.663	0.202	0.865
Server1	44654475	0.369	0.085	0.454
Server2	42749805	0.420	0.099	0.519
Financ1	36115196	0.222	0.083	0.305
Financ2	17693541	1.640	0.393	2.033
Web1	138040640	3.141	0.000	3.141
Web2	135387780	2.928	0.000	2.928

Table 3 Write savings achieved by 256 MB write buffer

Traces	Write savings (%)	
	Arbitrary buffer	Improved buffer
PC1	9.31	17.63
PC2	16.63	30.88
Server1	20.73	41.01
Server2	21.11	41.62
Financ1	29.10	54.67
Financ2	33.44	61.20

Table 4 Write savings achieved by 512MB write buffer

Traces	Write savings (%)	
	Arbitrary buffer	Improved buffer
PC1	10.67	19.51
PC2	21.10	35.00
Server1	22.90	42.37
Server2	24.12	43.73
Financ1	33.81	63.90
Financ2	36.86	68.10

6.3 Write buffer management

To reduce programs to flash chips, we employ a write buffer to filter some of the write requests. Write saving [12] is used to measure the performance achieved by write buffer. Write saving is the percentage of total writes prevented from reaching flash chips. If the host writes 100 pages to SSD, but only 60 pages are received by flash chips, the write saving is 40%. We explore two replacement policies for write buffer. The first is the arbitrary 4-way set-associative scheme, while the other is the improved 4-way set-associative scheme with the guidance of hot data-aware page table. Tables 3 and 4 present the write savings achieved by 256 and 512 MB write buffers, respectively. It is obvious that the improved buffer constantly outperforms the arbitrary one. As described in subsection 5.2, evictions in the improved write

Table 5 Comparison of two types of programs

Traces	Arbitrary program		Hot data-aware program	
	Erase blocks	Avg. valid pages	Erase blocks	Avg. valid pages
PC1	902959	7.41	851402	4.72
PC2	704565	0.96	691701	0.82
Server1	128175	30.11	93736	19.06
Server2	142574	30.08	106195	19.46
Financ1	1013722	35.11	978375	34.05
Financ1	69204	22.97	68387	22.61

buffer are mostly triggered by replacement of page table cache. First, the page table cache tends to evict cold entries. User data related to these cold entries are also cold. Second, hints supplied by 2-bit predictors inform write buffer which pages are cold. As the improved write buffer is aware of cold data, it obtains a higher write saving.

6.4 Hot data-aware program

LazyFTL distinguishes hot pages from cold pages, and writes them to separate super blocks. We refer to this intelligent behavior as hot data-aware program. Correspondingly, a method unaware of hot pages is called arbitrary program. We compare the two methods via simulations. Two metrics are defined to measure the performance. One is the number of blocks to be erased during garbage collection. This metric relates to the life span of P3Stor. The other is the average number of valid pages to be migrated when a block is recycled. This metric relates to the delay of garbage collection. Ideally, we expect these two quantities to be as small as possible.

The experimental results are given in Table 5. Generally, the hot data-aware program outperforms the arbitrary method, especially on Server1 and Server2. These two traces exhibit hybrid access patterns mixed with cold and hot pages. The arbitrary method is not competent for them. PC2 contains large chunks of overwrites only. There are no valid pages to migrate for garbage collection. Thus, the two methods behave similarly. Financial1 and Financial2 are write-intensive workloads. All pages are overwritten randomly and frequently. It is thus, not possible to optimize the programs for either of the methods. The hot data-aware program is good at scheduling write sequences with mixed cold and hot pages. These types of sequences are the most popular access patterns in enterprise-scale storage systems. As hot data-aware program migrates fewer valid pages and erases fewer physical blocks for garbage collection, LazyFTL responds rapidly to user's I/O requests. Moreover, P3Stor has a longer life span.

6.5 Lifecycle analysis

P3Stor uses three methods to extend its life span. The most important one is the write buffer. As explained in subsection 6.3, with a 512 MB write buffer, P3Stor almost reduces write traffic by 70% for some workloads. The prototype of P3Stor is deployed with a 2 GB write buffer, which can filter at least 50% of programs for most workloads. This means that the life span is almost doubled. The second method used to extend life span is the efficient garbage collection. As discussed in subsection 6.4, guided by the hot data-aware page table, the garbage collector erases fewer blocks. For typical enterprise-scale workloads mixed with hot and cold data requests as in Server1 and Server2, LazyFTL's garbage collection policy erases less blocks by 1.3 times compared with the arbitrary method. The last measure used to extend life span is the lazy wear-leveling. Since the cold data is stored in blocks that are nominally worn out, theoretically, all physical blocks are worn out by hot data. The nominally worn out blocks occupied by cold data are extensions of P3Stor's life span. Generally, the life span of P3Stor can be doubled or lengthened even further.

7 Conclusions

We have designed a parallel flash-based storage device called P3Stor that makes full use of flash chips. The architecture of P3Stor employs four levels of parallelisms. Module- and bus-level parallelisms are used to increase bandwidth, while chip-level parallelism is used to hide I/O latency. Furthermore, interface-level parallelism is used to support multiple concurrent transactions. Based on the proposed architecture, we designed LazyFTL to manage the address space. LazyFTL is able to distinguish hot data from cold data, which produces useful hints for garbage collection. Moreover, guided by hot data identification, the write buffer not only reduces the response time of write requests, but also significantly reduces programs to flash chips, which helps in extending P3Stor's life span. Besides that, LazyFTL reduces background operations by performing as little wear-leveling as possible. Performance evaluation by trace-driven simulations and theoretical analysis show that P3Stor achieves high performance and its life span can be more than doubled.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (Grants Nos. 60736013, 60903040, 61025009), and the Program for New Century Excellent Talents in University (Grant No. NCET-08-0145). We are grateful to Professor Weisong Shi for his valuable suggestions to improve this paper.

References

1 Samsung Electronics. K9WAG08U1A/K9K8G08U0A/K9NBG08U5A Flash Memory Datasheet

2 Boboila S, Desnoyers P. Write endurance in flash drives: measurements and analysis. In: Proc of the Eighth USENIX Symposium on File and Storage Technologies (FAST10). San Jose, CA, USA, 2010. 115–128

3 Caulfield A M, Grupp L M, Swanson S. Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications. In: Proc of ASPLOS'09. Washington, DC, USA, 2009

4 Seong Y J, Nam E H, Yoon J H, et al. Hydra: A block-mapped parallel flash memory solid-state disk architecture. IEEE Trans Comput, 2010, 59: 905–921

5 Chung T, Park D, Park S, et al. System software for flash memory: A survey. In: Proc of the International Conference on Embedded and Ubiquitous Computing, Seoul, Korea, 2006. 394–404

6 Lee S, Park D, Chung T, et al. A log buffer based flash translation layer using fully associative sector translation. IEEE Trans Embed Comput Syst, 2007, 6: 18

7 Park S Y, Cheon W, Lee Y, et al. A re-configurable FTL (flash translation layer) architecture for NAND flash based applications. In: Proc of International Workshop on Rapid System Prototyping, Paris, France, 2007. 202–208

8 Cho H, Shin D, Eom Y. KAST: K-associative sector translation for NAND flash memory in real-time systems. In: Proc of Design, Automation and Test in Europe (DATE09), Nice, France, 2009. 507–512

9 Gupta A, Kim Y, Urgaonkar B. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings. In: Proc of the 14th international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS09), Washington, DC, USA, 2009. 229–240

10 Chen F, Luo T, Zhang X D. CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives. In: Proc of the Ninth USENIX Symposium on File and Storage Technologies (FAST11). San Jose, CA, 2011

11 Gupta A, Pisolkar R, Urgaonkar B, et al. Leveraging value locality in optimizing NAND flash-based SSDs. In: Proc of the Ninth USENIX Symposium on File and Storage Technologies (FAST11). San Jose, CA, 2011

12 Soundararajan G, Prabhakaran V, Balakrishnan M, et al. Extending SSD lifetimes with disk-based write caches. In: Proc of the Eighth USENIX Symposium on File and Storage Technologies (FAST10). San Jose, CA, 2010. 101–114

13 Hennessy J, Patterson D. Computer Architecture—Quantitative Approach. 3rd ed. San Fransisco: Morgan Kaufmann, 2003

14 Reed I S, Solomon G. Polynomial codes over certain finite fields. SIAM J Appl Math, 1960, 8: 300–304

15 Chang L P, Kuo T W. An adaptive stripping architecture for flash memory storage systems of embedded systems. In: Proc of IEEE Eighth Real-Time and Embedded Technology and Applications Symposium. San Jose, CA, USA, 2002. 178–196

16 Wu C H, Chang L P, Kuo T W. An efficient R-tree implementation over flash-memory storage systems. In: Proc of the 11th International Symposium on Advances in Geographic Information Systems. New Orleans, Louisiana, USA, 2003. 17–24

17 PC traces for flash memory research. http://newslab.csie.ntu.edu.tw/flash/index.php?Selected Item =Traces

18 OLTP Trace from UMass Trace Repository. http://traces.cs.umass.edu/index.php/Storage/Storage

19 Websearch Trace from UMass Trace Repository. http://traces.cs.umass.edu/index.php/Storage/Storage

20 Kim Y, Tauras B, Gupta A, et al. FlashSim: a simulator for NAND flash-based solid-state drives. Technical Report, CSE-09-008 of the Pennsylvania State University. 2009

21 Belady L. A study of replacement algorithms for a virtual-storage computer. IBM Syst J, 1966, 5: 78–101