Journal of Systems Architecture 57 (2011) 354–365

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

A comprehensive study of energy efficiency and performance of flash-based SSD ☆

Seonyeong Park [a], Youngjae Kim [b], Bhuvan Urgaonkar [c], Joonwon Lee [d], Euiseong Seo [e,*]

[a] Division of Computer Science, Korea Advanced Institute of Science and Technology, Daejeon, Republic of Korea
[b] National Center for Computational Sciences, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA
[c] Department of Computer Science and Engineering, The Pennylvania State University, University Park, PA 16803, USA
[d] School of Information and Communication Engineering, Sungkyunkwan University, Suwon, Republic of Korea
[e] School of Electrical and Computer Engineering, Ulsan National Institute of Science and Technology, Ulsan, Republic of Korea

ARTICLE INFO

Article history:
Received 7 February 2010
Received in revised form 13 December 2010
Accepted 30 January 2011
Available online 25 February 2011

Keywords:
Flash memory
SSD
Energy
Power
Filesystems
Storage

ABSTRACT

Use of flash memory as a storage medium is becoming popular in diverse computing environments. However, because of differences in interface, flash memory requires a hard-disk-emulation layer, called FTL (flash translation layer). Although the FTL enables flash memory storages to replace conventional hard disks, it induces significant computational and space overhead. Despite the low power consumption of flash memory, this overhead leads to significant power consumption in an overall storage system. In this paper, we analyze the characteristics of flash-based storage devices from the viewpoint of power consumption and energy efficiency by using various methodologies. First, we utilize simulation to investigate the interior operation of flash-based storage of flash-based storages. Subsequently, we measure the performance and energy efficiency of commodity flash-based SSDs by using microbenchmarks to identify the block-device level characteristics and macrobenchmarks to reveal their filesystem level characteristics.

1. Introduction

NAND flash memory (henceforth referred to as flash memory) is widely used in a diverse range of computing systems as storage devices, from hand-held scale embedded systems to enterprise-scale high performance servers, as a replacement for traditional hard disks.

In comparison to hard disks, flash memory has many characteristics suitable for use as the storage medium of choice in various fields. First, its small form-factor helps its placement in small devices. In addition, flash memory exudes no noise or vibration, along with kinetic shock resistance because it uses no mechanical components. Finally, it provides fast read speed. Because it does not require any moving parts to access data, random read operations are as fast as sequential reads.

Energy efficiency, as well as performance, of the storage subsystem is very important in mobile devices. Laptop hard disks, whose power consumptions has already been reduced from that of the desktop hard disks, consume about 2 W while in operation. In contrast, the power consumption of flash SSDs during operation is in the range of few hundred milliwatts (mW), and it consumes only a couple of microwatts (μW) when idle.

However, in spite of flash memory's advantages, flash memory has an important limitation. As opposed to the traditional storage media, the data that is already written on flash memory cannot be updated immediately. The data has to be erased before rewriting. Moreover, the unit size of an erase operation is bigger than the unit size of a read/write operation. To overcome this gap, flash memory-based storage systems employ an emulation layer called the FTL (flash translation layer) [10], which emulates the normal block devices, such as hard disks.

Flash-based SSDs are the storage devices that use flash memory as their storage media, and employ the FTL to emulate normal hard disks. Because they can be used without modifying existing operating systems and applications, flash-based SSDs are now widely used in laptops and mobile embedded systems [32].

Although the FTL enables flash memory storage devices to replace conventional hard disks, it usually requires a significant amount of fast and non-volatile memory such as battery-backed DRAM to store the translation mapping table for the FTL, along with an embedded processor, which should be powerful enough to search and manipulate the mapping data quickly. Consequently, the FTL increases power consumption by the embedded processor and memory included in the storage devices.

Research has been undertaken to investigate the performance of flash memory and storage devices using flash memory in embedded

☆ This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2009-0089491 and 2010-0003453).
* Corresponding author.
E-mail addresses: parksy@calab.kaist.ac.kr (S. Park), kimy1@ornl.gov (Y. Kim), bhuvan@cse.psu.edu (B. Urgaonkar), joonwon@skku.edu (J. Lee), euiseong@unist.ac.kr (E. Seo).

1383-7621/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.sysarc.2011.01.005

systems [4]. However, there has been relatively little effort expended to improve the energy efficiency of flash memory storage devices.

In order to provide the insight to improve both performance and energy-efficiency of flash-based storage systems, this paper analyses the performance and energy efficiency of the flash-based storage systems at diverse levels of system hierarchy, which spans from the FTL to filesystems.

First, we observe the computational and spatial overhead of a few popular FTLs under various workloads using a flash SSD simulator, which was built for our previous research [12]. By using this simulator we can measure the computational overhead, data transfer overhead, and memory usage of SSDs for specific workloads. The information obtained from this analysis is intended to help storage system developers to design power-aware storages that fit their target workload.

Next, we empirically analyze the energy efficiency and performance of SSDs in the market. We measure the performance and the energy efficiency according to block level I/O patterns. In addition, we observe the changes in the energy efficiency and performance according to the filesystem and workload combinations using a macrobenchmark tool on two different filesystems. This empirical analysis will provide valuable information for system software engineers to design filesystems and applications that fits the power and performance characteristics of flash-based storage devices because software engineers are not able to modify the internals of storage devices and have to use off-the-shelf products most of the time.

The remainder of this paper is organized as follows; Section 2 provides background about flash memory and the FTL. An in-depth analysis of FTL overhead is presented in Section 3 and an empirical study of the energy efficiency and performance characteristics of flash-based SSDs is presented in Section 4. We conclude our work in Section 5.

2. Background

This section introduces the diverse characteristics of flash memory and analyzes the mechanisms and pros and cons of popular FTL algorithms.

2.1. Flash memory storage

A flash memory chip is a non-volatile memory component, which can read and write data electronically. Because flash memory is purely electronic, it shares some positive characteristics with DRAM. For example, it does not generate any noise and vibration. Also, it is more resistant to kinetic shocks than hard disks. However, it also has significant differences from DRAM, especially in the interface.

While each byte or word is addressable in a DRAM component for both read and write operations, the basic unit of read and write operations for flash memory is a page. Usually, a page is contiguous memory space, with a typical size from 512 B to 4 KB, and if a page has been already written to, it must be erased before it is written to again. However, the unit of the erase operation is different from that of the read and write operations. A block, which is the unit of an erase operation, usually consists of multiple (from 32 to 128) pages. Because most of the filesystems currently in active use have been built on the assumption that overwriting does not create any special concerns, an emulation layer is required to manage the overlying flash memory and expose traditional storage device interfaces externally.

In addition to the interface gap, the time needed for a read operation, a write operation, and an erase operation, differs from each other. Usually, as shown in Table 1, a read operation takes a significantly shorter time to finish than a write operation. Also, the time for an erase operation is about one hundred times longer than that of a read operation. Consequently, an in-place rewriting approach, which rewrites a whole block after erasing the block for rewriting a page in the block, is impractical because of its enormous temporal overhead. For this reason, while emulating the writing or overwriting operations, the emulation layer should choose a free page, which is to be mapped to a logical page so that it minimizes the erase operations.

Flash-based SSDs are storage devices which use flash memory as their storage medium and equip FTLs to manage the flash memory. By employing a standard interface such as S-ATA, SAS or E-IDE, SSDs are externally identical to normal hard disks. They also work like conventional hard disks. They are attached to the host computers via standard interfaces such as S-ATA ,SAS or EIDE, and they share the same command set as hard disks. Consequently, they can be used in existing systems, instead of the traditional hard disks, without any modifications. As a result, they are currently the most viable option to replace hard disks for flash memory storage devices.

Although energy consumption of flash memory is very low even when it is working as listed in Table 1, there are many components inside a flash-based SSD which can affect the power consumption of a flash-based SSD as illustrated in Fig. 1.

The embedded processor executes the system management code and controls the other components. Unlike hard disks, which normally use their dedicated ASICs, flash-based SSDs use general-purpose embedded processors [18], which have high performance as well as high power consumption because the system operating codes of SSDs such as the FTL are significantly more complex than that of the hard disks. Therefore, it can easily be expected that both the complex management scheme and the high performance general purpose embedded processor will increase power consumption significantly.

The complex management code enforces the use of not only the high performance processor, but also a large amount of DRAM, which is used to accommodate management program code, dynamic data, as well as data cache. Generally speaking, the memory subsystem is the second largest power consumer in a computing system and the power consumption by the memory subsystem increases depending on its capacity [28]. Therefore, the large amount of DRAM equipped in a flash-based storage device is also expected to consume a significant amount of power.

Both the host interface controllers such as SATA or SAS, and the internal buses between the flash memory controller and the host interface consume a certain amount of leakage power like embedded processors when they are idle. They consume dynamic power only when they transfer data or requests. The host interface controllers and internal buses of SSDs are basically the same as that of HDDs. Also, the number of requests and the amount of data that flow through them are the same. Therefore, we do not consider the power consumption by the host interface and internal buses in this paper.

Finally, a FTL maintains at least a translation mapping table, whose size varies from hundreds of KB to tens of MB depending

Table 1
Performance and energy characteristics of NAND flash memory using 2 KB page/128 KB block [12,21].

Access time	Read	130.9 us/page
	Write	405.9 us/page
	Erase	1.5 ms/block
Energy consumption	Read	4.72 uJ/page
	Write	38.04 uJ/page
	Erase	527.68 uJ/block

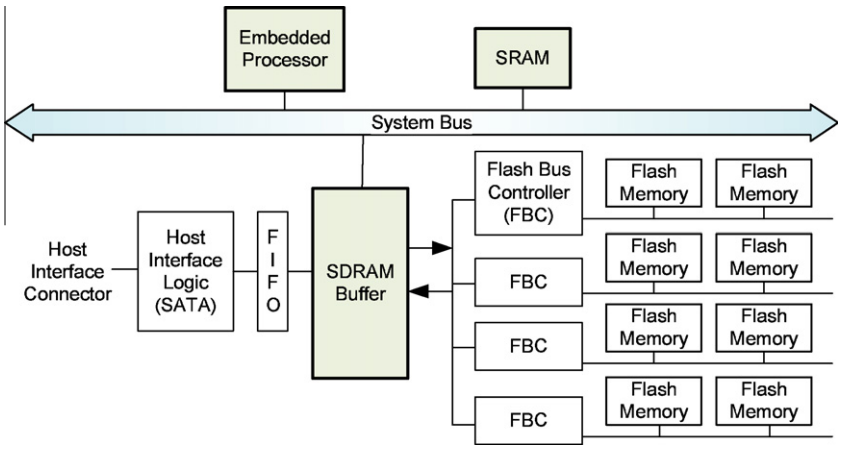


Fig. 1. Block-diagram of a typical SSD inside [3].

on the underlying FTL algorithms and implementation details. This mapping table is accessed in every request and should be maintained when the storage system is powered off. Therefore, the memory for this translation mapping table should be stored in fast and non-volatile media such as battery-backed DRAM or battery-backed SRAM. Because battery-backed DRAM is also a significant power consumer, the memory component for storing the mapping table is also suspected to contribute to the power consumption of flash-based SSDs

2.2. Flash translation layer

The FTL maintains a mapping table of virtual sector numbers, which are seen from the host system, to physical sector numbers of the flash memory. It emulates the functionality of a normal block device by exposing only read/write operations to the upper software layers and hiding the presence of the erase operation.

Processing read requests in the FTL is simple. When the read request occurs from the host system, the FTL returns the corresponding physical sector number after searching the mapping table.

In comparison to processing read requests, write requests induce a complex update process. Basically, the sequence of processing write requests is as follows; (i) find a suitable erased page to be written, (ii) write the new data over the chosen page, (iii) invalidate the original page mapped to the logical page currently in process, if it exists, and (iv) update the mapping table to reflect this change.

FTL algorithms can be roughly categorized into three groups by their mapping granularity.

The page-mapping algorithm [16], which is a basic form of the FTL, uses a translation table, which translates external logical page addresses into internal physical page addresses on the flash memory as shown in Fig. 2(a). Because the granularity of each entry is a page, each mapping entry implies an one-to-one mapping relationship from a logical page to the corresponding physical page. When the overwriting operation occurs, the FTL only needs to remap the corresponding entry.

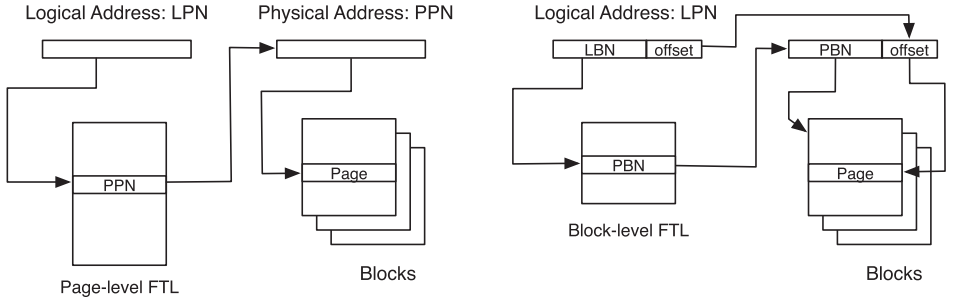
By using this straightforward table management, the page-mapping algorithm has a fast translation speed and simple page-update mechanism, which only changes the mapping entry for the page being updated. However, because there should be as many mapping entries as the number of pages, the mapping table takes a large amount of space, which we will explore in Section 3.3.

Therefore, the block-mapping algorithm [29] and its variants were proposed. As shown in Fig. 2(b), the mapping table in the block-mapping FTL only contains the block granularity mapping entries, each of which maps a logical page address to the combination of a physical block number and an offset number. The actual physical page, which stores the data for the logical page, is determined by the offset field of the logical page addresses in the block.

Although the table size of the block-mapping FTL is very small, processing partial rewrites of previously written blocks induces a substantial amount of copy and erase operations. This is because, in order to overwrite a page, a new free block has to be allocated, and all the pages, except the page to be overwritten in the original block, have to be copied over to the pages with the same offsets in the new block.

Therefore, the hybrid FTL (also known as the log-based FTL) [19,23], illustrated in Fig. 3 has gained popularity [17] in commodity flash-based storage devices, of which production cost is of the highest priority.

The hybrid FTL has a few spare blocks, called log blocks, which are used as temporary space for the updated pages. When a page overwrite operation occurs, a free page in the log blocks is



(a) Page-level Address Translation Scheme (b) Block-level Address Translation Scheme

Fig. 2. Address translation in the page-mapping and block-mapping FTL.

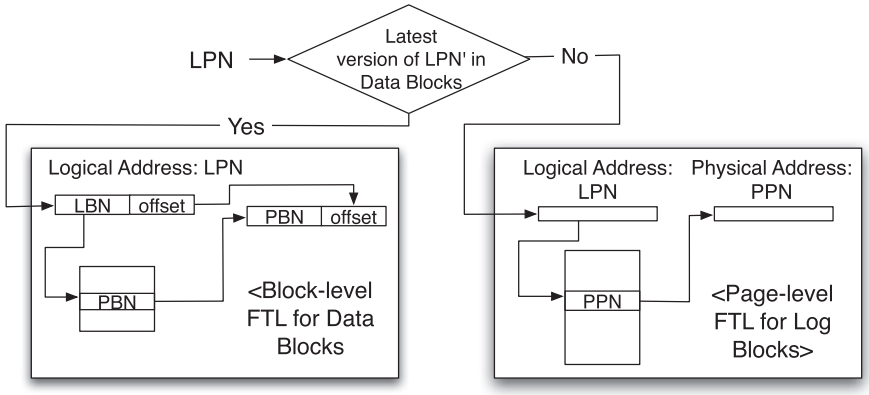


Fig. 3. Address translation in the hybird FTL.

allocated for the updated page and the original page is marked as an invalid page. Thus the hybrid FTL has two mapping tables. One is block granularity mapping, which is the same as that of the block-mapping FTL, and the other is page granularity mapping for pages in the log blocks. By using this mechanism, the hybrid FTL saves a lot of space for the mapping tables and achieves a fast overwriting speed.

In general, log blocks can be as large as 5% of total storage capacity [22,24]. When write requests occur continually, eventually there will be no available free blocks left in the log block area. In order to extract free log blocks from used log blocks, garbage collection operations are conducted. Garbage collection merges invalidated blocks or partially invalidated blocks into a smaller number of valid blocks so that they can be used as log blocks after erase operations. Garbage collection is initiated on demand when the amount of free log blocks drops below predefined thresholds or when the storage systems are in idle to prepare for burst write requests. By employing the garbage collection mechanism throughput for burst write phases improve significantly by securing enough log blocks in advance.

Massive overwriting operations generate a large amount of partially invalid blocks and consequently induce the shortage of free pages in the log blocks. Therefore, the partially invalidated blocks should be merged into fewer fully valid blocks, so that the partially invalidated blocks can be erased to be used as free blocks and free log blocks. This is called a merge operation. There are four types of merge operations, which are illustrated in Fig. 4.

In Fig. 4(a), log block B contains all valid, sequentially written pages corresponding to data block A. In this case, a simple *switch merge* is conducted, whereby log block B becomes new data block and the old data block A is erased. During the switch merge the log-block mapping entries are erased and the block mapping entry is changed to refer to the new log block so that the log block becomes a regular data block.

When only part of a block is invalidated by overwriting and the overwritten pages are located in the same block sequentially as shown in Fig. 4(b), just switching the mapping entry does not produce any free blocks. By copying the remaining valid pages in the original block to the log block, the *partial merge* turns the original block into a fully invalid one to be safely erased. In the partial merge operation, every procedure after copying the remaining pages is the same as the switch merge.

A *full merge* operation involves the largest overhead among the three types of merges. As illustrated in Fig. 4(c), log block B is selected as the victim block. The valid pages from log block B and its corresponding data block A will be copied into a newly erased block C, and block A and B will be erased.

The partial merge and full merge, which take a significant time to finish, primarily occur due to small random writes. Therefore, regardless of the type of the FTL, random writes on flash-based

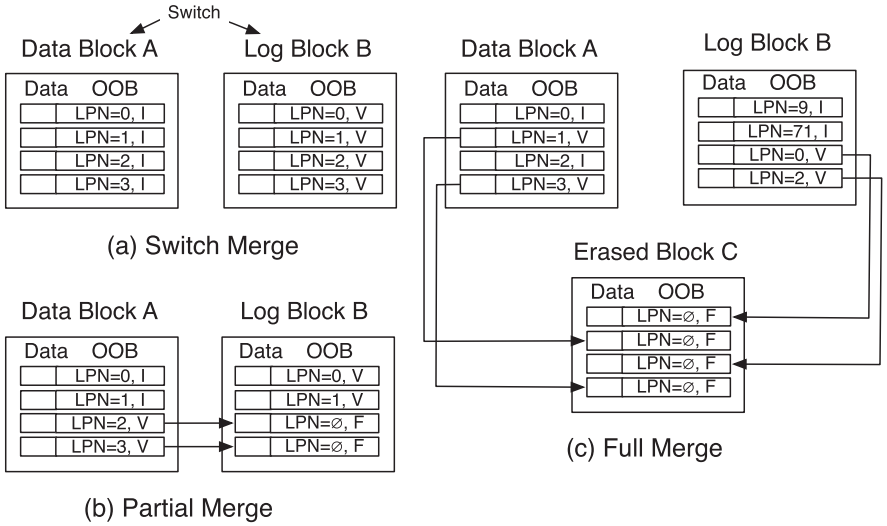


Fig. 4. Three kinds of merge operations in hybrid FTLs [19].

SSDs generally show significantly slower throughput than sequential writes [5].

3. Analysis of FTL overhead

3.1. Simulation environment

In our previous research [12], we developed a flash-based storage simulator based on *DiskSim 3.0* [6]. We enhanced the simulator to simulate the power consumption model of a flash-based SSD based on the flash memory specification described in Table 1. The capacity of the target SSD is 1GB.

We measure the computational overhead with the simulator while running two real-world workloads described in Table 2. *OLTP* (online transaction processing) is a filesystem trace collected from the financial system. The OLTP workload is primarily composed of small random writes. The average I/O size is 4.38 KB and 91% of requests are write operations. Among all the requests, 98% of requests change the offset from the resulting offset of the last request, which means a random request. The next trace, *OLAP* (online analytical processing), is a trace extracted from a server running the *TPC-H* benchmark suite [30]. In contrast to the OLTP trace, it primarily consists of small random read requests.

Our simulator simulates three different kinds of FTL schemes; page-mapping FTL, FAST (Fully Associative Sector Translation) [23], LAST (Locality-Aware Sector Translation) [22] and DFTL (Demand-based Page-mapped FTL) [12]. FAST and LAST are variants of hybrid FTLs, and DFTL is a FTL algorithm designed for large-scale flash-based storage devices.

FAST allows log blocks to be shared by all data blocks. This improves the utilization of log blocks as compared to the conventional hybrid FTLs. FAST keeps a single sequential log block dedicated for sequential updates while other log blocks are used for performing random writes. Thus, it cannot accommodate multiple sequential streams and does not provide any special mechanism to handle temporal locality in random streams.

The more recently developed LAST scheme tries to alleviate the shortcomings of FAST by providing multiple sequential log blocks to exploit spatial locality in workloads. It further separates random log blocks into hot and cold regions to reduce full merge cost. In order to provide this dynamic separation, LAST depends on an external locality detection mechanism. However, the authors of LAST realized that the proposed locality detector cannot efficiently identify sequential writes when the small-sized write has sequential locality. Moreover maintaining sequential log blocks using a block-based mapping table requires the sequential streams to be aligned with the starting page offset of the log block in order to perform switch-merge. Dynamically changing request streams may impose severe restrictions on the utility of this scheme to efficiently adapt to workload patterns.

The capacity of a log-block area affects write performance depending on workload patterns. The relationship between log-block capacity and performance is well studied by Lim et al. [24]. In general larger log-block areas have higher manufacturing cost, but have the advantage of faster write response times. According to the evaluation results from the inventors of LAST, log blocks as large as approximately 2% of total capacity is sufficient to have performance benefit [22]. In our simulator, FAST and LAST commonly have 50MB of log blocks, which is approximately 5% of the total capacity. Observing the workload patterns for our evaluation, we conclude that log block capacity as large as 5% of total capacity is large enough to benefit from temporal locality.

In SSDs that employ FTLs with log-blocks, actual writing operations to permanent locations occurs during garbage collection. Performance drops during garbage collection operations may not affect the user experiences because they are usually done during idle time. However, the energy consumption from merge operations contributes to the overall energy efficiency of storage systems. Therefore, in order to analyze energy efficiency, we have to count energy consumption for merge operations in garbage collection. When our simulator measures the cost for processing workloads, it considers the energy consumption during workload handling, as well as during garbage collection after finishing workloads.

DFTL makes use of the presence of temporal locality in workloads to judiciously utilize the small on-flash SRAM. Instead of the traditional page-mapped FTL approach of storing all the address translation entries in the SRAM, it dynamically loads and unloads the page-level mappings depending on the workload access patterns. Furthermore, it maintains a complete image of the page-based mapping table on the flash device itself. This enables DFTL to exploit the locality of workload patterns by caching frequently accessed mappings in the SRAM and storing other huge mapping entries on flash memory.

We compare the characteristics of these four FTLs in terms of performance and energy.

3.2. Run-time overhead

Although the power consumption of flash memory is relatively small in comparison to the other components, erasing and writing flash memory, which take significantly greater energy than reading, could affect the system's overall energy efficiency. Thus, we estimate the energy consumption of each kind of flash memory operation during the workload execution. Fig. 5 shows the results.

Only some parts of page-mapping entries, which are frequently used, are accommodated in RAM and the selection of the entries to be stored in RAM is done on the fly based on the workload changes. Therefore, the overhead for managing the address translation table occurs only with DFTL. Fig. 5 shows the number of data writes, address translation writes for evicting entries as well as the number of data reads, address translation reads for loading entries.

The page-mapping FTL does not require merge operations because it allocates space at page granularity. Consequently erase operations from conducting merge operations do not occur in the page-mapping FTL. As a result, a minimal amount of erase operations is necessary. Because erase operations consume significantly more energy than the other operations, the page-mapping FTL is expected to consume less energy for flash memory operations than the other FTLs.

Using log blocks reduces copy and erase operations by deferring merge operations and aggregating write operations in comparison to the conventional block mapping algorithms. Therefore, the number of flash operations ,such as read, write and erase operations, decreases by employing log-block schemes.

As expected, simulation results illustrated in Fig. 5 reveal that FAST consumes significant energy for both erase and write operations during merge operations in the garbage collection phase. Also, we verified that out of all the FTLs, the page-mapping FTL consumes the least amount of energy.

Without a merge operation, the time to process read or write requests in the FTL is usually finished in a discrete time period. However, a merge operation takes a much longer time than read or write requests, because it has to find appropriate victim blocks

Table 2
Workload characteristics used in our simulation.

Workload	Avg. Req. size	Rd.:Wr.	Randomness	Inter-arrival time	Simulated time
OLTP [1]	4.38 KB	9.0:91.0	98%	133.50 ms	11.97 h
OLAP [14]	12.82 KB	95.0:5.0	82%	155.56 ms	10.29 h

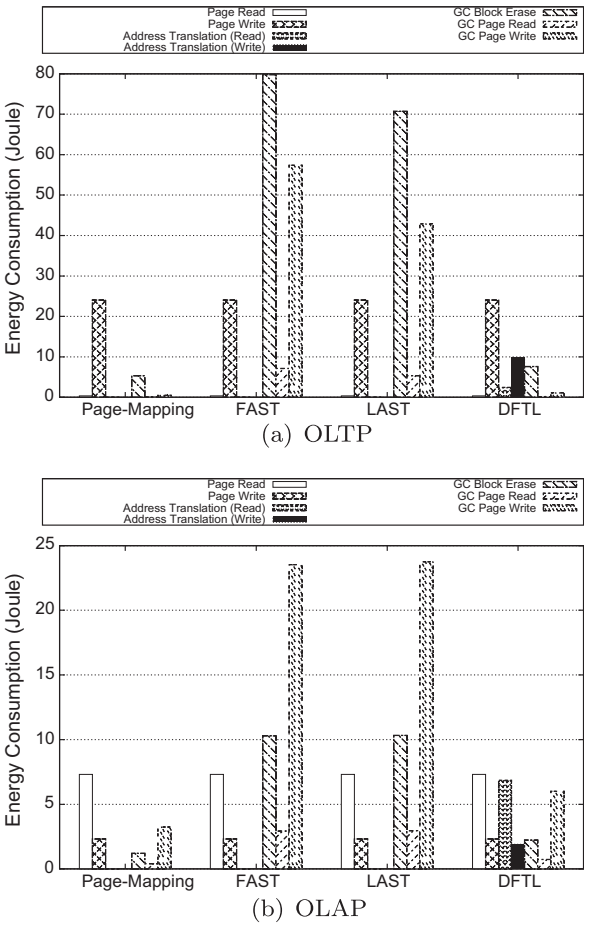


Fig. 5. Energy consumption by each flash memory operation after running the workload under the three different FTLs.

to be merged by searching the mapping table along with other metadata. Naturally, a merge operation is expensive from an energy usage point of view.

Fig. 6 shows the average response time of all the requests in the workload and the number of total searches in all the merge operations that occurred during execution.

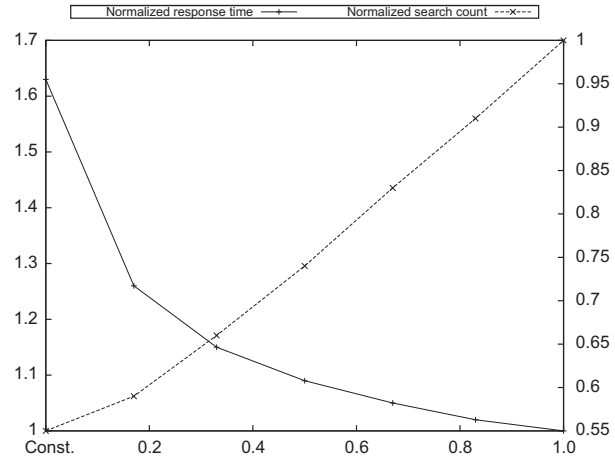


Fig. 6. Normalized average response time and number of FTL table search operations for all the merge operations during the workload execution according to the victim search window size of merge operations in DFTL. "Const." means choosing a victim block randomly. "1.0" means searching for the best candidates among all the partially (or fully) invalidated blocks.

If optimized victims are chosen for a merge operation, the result yields more free blocks than a merge operation with sub-optimal victims. Therefore choosing good victims reduces the number of merge operations and, naturally, the response time for random write operations. However, in our simulation, we find that the effort to find good victims, which increases the search operations for a merge operation, exceeds the effort saved from avoiding future searches in total.

The search operation induces energy consumption in the processor and system bus. Consequently, we can conclude that minimizing victim search operations will save energy consumption for merge operations, although it harms the response time of the storage device by creating more partially invalidated blocks than searching over more candidate blocks.

3.3. Memory space usage

FTLs require fast access memory, such as DRAM, in order to accommodate translation mapping entries, and DRAM consumes a significant amount of power while it is powered-on and ready to serve requests. However, it is hard to provide absolute numbers of energy consumption from the memory subsystem of SSDs because both the memory access timing parameters and energy requirements for an access operation are determined by design and fabrication technology of the memory. In this section, we analyze the memory requirement and memory access count of each FTL scheme. We believe that this analysis results will provide meaningful clues to understand the energy and power consumption patterns of memory subsystems in SSDs.

The power consumption of DRAM is usually as high as a few hundreds of mW per chip with ordinary SDRAM technology [2] and around a hundred mW per chip with LPRAM technology [11].

There is a slight increase in power consumption as big as a few tens of mW while access operations are being served [2][11]. However, because the actual service time for requests is extremely short in comparison to the total operation time, the energy consumption of the memory system mostly depends on the size of equipped memory. Also, it is revealed that the actual dynamic power is significantly less than that on the published specification [27].

Considering that the time to access DRAM memory for retrieving a translation entry is around a few ns (nanoseconds), we expect that the contribution to the overall system energy consumption from handling read requests is negligible, and that capacity is a dominant factor for power consumption of the memory subsystem when read requests greatly outnumber write requests. Therefore, we analyze the memory capacity requirement by each FTL algorithm.

Fig. 7 shows the memory space of the FTL mapping table of each FTL algorithm, as well as the generic block-mapping algorithm for comparison. It can be interpreted as the power consumption of memory components during idle time or processing read requests.

Each mapping entry is assumed to be 8 bit aligned, which means that the smallest data unit for the mapping entry is a byte. For example, a 512 MB flash storage consists of 262,144 2 KB-pages. That number can be addressed with a 24-bit variable, and therefore the total size of the mapping table will be $262,144 \times 3 = 786432$ bytes, which is 768 KB.

As shown in Fig. 7, as the capacity of the storage device grows, the size of the mapping table also grows in all algorithms. However, the memory requirement of the page-mapping FTL increases much faster than that of the other algorithms. Therefore, it is expected that the power consumption in idle of the large capacity flash-based storages will consume more power with the page-mapping FTL than the block-mapping FTL.

In our analysis, both FAST and LAST use 5% of total capacity as log block space. The global mapping tables of both FAST and LAST

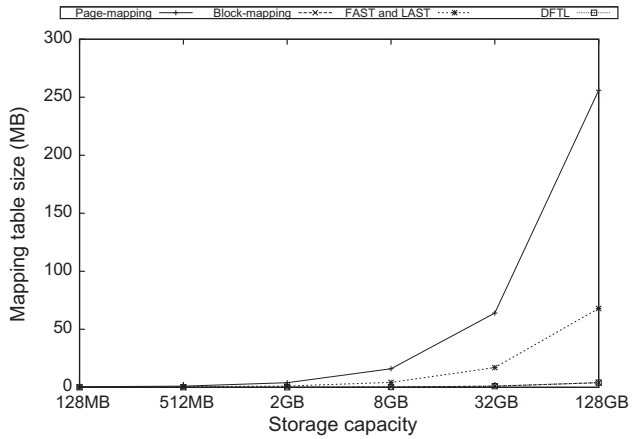


Fig. 7. Size of the mapping tables in the two representative FTL schemes.

are the same as that of the block-mapping FTL. Therefore, the difference in memory requirements arises only at the log-block mapping tables. Generally, log-blocks are managed by page-granularity mapping tables, and the memory requirement for managing log-blocks increases as the size of log-block area grows as shown in Fig. 7.

Basically, DFTL requires the same amount of memory as the block-mapping FTL, except for the memory that manages the page-mapping area, which exists to boost performance. Read and write performance under DFTL depends both on spatial locality of workloads and the size of the page-mapping area. If a workload has high spatial locality, even a small page-mapping cache will enhance throughput significantly. To contrast, if a workload has low spatial locality, the performance improvement will be proportional to the size of the cache area. Existing research insists that approximately 32 KB is enough to benefit from the locality of general workloads [12]. Therefore, we assume that the cache size is fixed to 32 KB regardless of total capacity.

Actually, because most modern general purpose processors use 32-bit aligned addressing, commodity products typically use 32 bit entries for simplifying design and achieving high performance. So, our model to achieve Fig. 7 is thought to be conservative and, naturally, we can tell that the real sizes of the FTL tables, as well as the differences between those of the two FTL schemes are larger than those in the real world.

The hybrid FTL maintains the same block-granularity mapping table as the block-mapping FTL. Although the size of the mapping table for the pages in the log blocks depends on the internal implementation of the hybrid FTL, the size of the mapping table in the hybrid FTL would be similar to that of the block-mapping FTL. Also, the cached mapping table, which is a battery-backed DRAM or SRAM area in the DFTL, is comparatively small within the block-mapping FTL. This means that the power consumption in idle would be low under both the hybrid FTL and DFTL.

Frequent memory accesses occur when finding victim blocks for merge operations during write request processing. Although the access time is short and the power difference between access and idle is a few mW, the power consumption from massive memory accesses may sum up to a significant value.

The number of memory accesses for finding victim blocks depends on how many blocks a FTL algorithm scans over in search of a victim block. As we have already seen in Fig. 6, better performance through the use of larger scanning regions carry with it more memory accesses, which results in more energy consumption by the memory subsystem.

To summarize, a FTL algorithm trades the efficiency of merge operations, which is determined by the number of clean blocks out of partially invalid victim blocks, for the number of memory accesses to find the victim blocks.

4. Empirical analysis

The performance of a storage device that users actually feel is not the pure block-device-level I/O performance of the device, but the final result that comes from a combination of storage devices, filesystems and workloads. Therefore, in order to give users the best experience out of a storage device, the filesystem has to understand the characteristics of both the storage device and target workload, and utilize the storage device efficiently based on this understanding.

The performance of commodity SSDs were previously evaluated through block-level benchmark tools and workload emulators [9]. However, the existing analysis was only for performance, and not for power consumption or energy efficiency of SSDs. In this paper, we evaluate four commodity SSDs with various characteristics in terms of performance, power consumption and energy efficiency at both block level and filesystem level. Also, based on the evaluation results, we infer the internal configurations of the SSDs and draw some hints to obtain better performance and energy efficiency out of commodity SSDs.

To reveal the block device level performance as well as the combinatory performance of the storage device with filesystem and workload, we evaluate its performance and energy-efficiency while running both microbenchmarks and macrobenchmarks. The microbenchmark suite issues diverse patterns of block read/write operations and measures working time and power consumption. The macrobenchmark suite evaluates two different filesystems, ext2 [7] and LFS (the log-structured filesystem) [26], on each storage device used in our study.

4.1. Experimental setup

In our empirical study, we use three different SSDs and a HDD to investigate the performance and power characteristics from different hardware configurations. Table 3 presents their specifications. The throughput values in the table are not measured, but are from the vendor specifications.

OLD is one of the first-generation SSDs for the consumer market. It was released to the market at year 2006. MLC is a MLC flash-based SSD introduced in 2008. MLC flash has poorer performance and a shorter lifetime than SLC flash but it is less expensive. Due to multi-channel technology which enables reading from multiple flash memory chips simultaneously, SSDs with MLC flash have comparable performance in read operations [8], thus they are becoming popular in the mass market of SSD products. SLC is one of the high-end SSDs with high sustained throughput, which is being marketed as an enterprise-class SSD. By employing a parallelized controller for the SLC flash memory, it shows high performance for both read and sequential write. TGN is the state-of-the-art consumer-grade SSD, which was released in 2010. TGN is one of the third generation SSDs that can read and write at over 200 MB/s. For a comparison with a traditional storage device, we also evaluate a laptop hard disk denoted as HDD in Table 3. HDD is a laptop hard disk which consumes less power than the desktop or server hard disks.

Power consumption is measured with a Signametrics SM2040 Digital PCI Multimeter, which is able to acquire power values once a millisecond. We measured the power from +5 V and +12 V power lines in SATA power cables.

For the microbenchmark, we implemented a workload generator called *DIO Tool* to measure the performance and power consumption for various patterns of I/O operations. It uses a direct

Table 3
Specifications of the storage devices used in our work.

	OLD	MLC	SLC	TGN	HDD
Model	FSD32GB25M	1C32G	MSP7000	MMCRE28G5	WD1600BEKT
Vendor	Super talent	OCZ	MTron	Samsung	Western digital
Form factor	2.5 in.	2.5 in.	2.5 in.	2.5 in.	2.5 in.
Flash type/RPM	SLC	MLC	SLC	MLC	7200
Capacity	32 GB	32 GB	16 GB	128 GB	160 GB
Rd./Wr. Perf. (MB/s)	60/45	143/93	120/90	220/200	NA/NA

I/O interface provided by the Linux kernel to measure throughput without any effect from the operating system buffer cache. For the macrobenchmark, we test two filesystems, ext2 and LFS to study power consumption differences due to the overlying filesystems. We use an open-source LFS implementation [15].

4.2. Microbenchmark

Idle power, along with working power, is an important factor because it affects the battery life of mobile electronics. We measure the power consumption of each device for 10 min without sending any requests, as shown in Table 4.

The power consumption of OLD and SLC in an idle state is similar to or higher than that of HDD. TGN consumes the least power. Although this sample set is too small to generalize, we can easily see the tendency that a newer SSD consumes smaller idle power.

Considering that SLC and MLC are introduced in the same year, equipped memory size seems to be one of the major reasons of the difference between idle power of them. With few simple experiments, it is verified that SLC has larger buffer memory than MLC and SLC employs page-mapping FTL variants while MLC uses hybrid FTLs.

When a request larger than the buffer size arrives, the processing time dramatically increases from the time for a request smaller than the buffer size. By exploiting this characteristic we can identify the size of write buffer externally. By conducting simple experiments, we verify that SLC has approximately 16MB of write buffer, while MLC barely shows the performance benefit from write buffering. We also observe that it has very poor performance when writing data randomly. From this, we suspect that MLC equips small write buffer and this aggravates merge overhead under random writes.

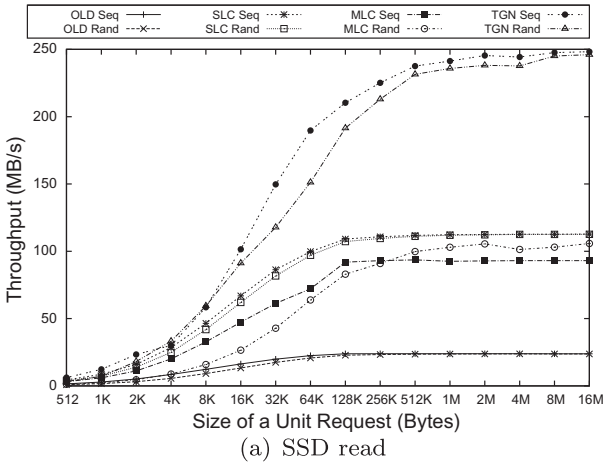
Most modern hard disks commonly provide low power modes, in which they stop spinning and consume minimal power. The sleep is initiated by delivering *ioctl* command for sleep to disks. However, all SSDs in our evaluation do not react to the sleep command. They keep the same idle power shown in Fig. 4 after issuing the sleep command. By this experiment, we can conclude that most commodity SSDs do not provide the low power idle mode, which is a common feature for hard disks.

To measure performance and power consumption characteristics, we employ four categories of I/O operations; sequential read, random read, sequential write, and random write. In each category, we change the request size.

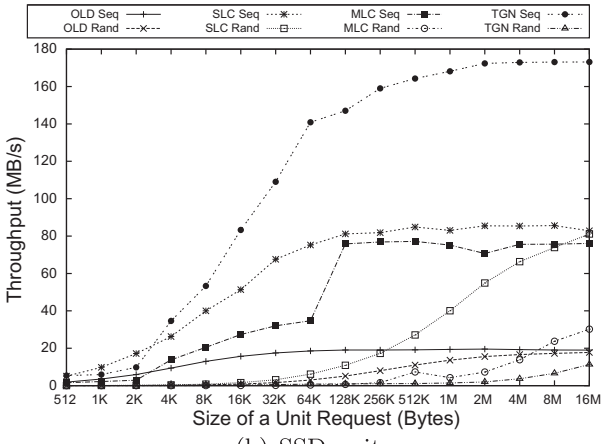
The throughput of HDD has little difference between reads and writes as shown in Fig. 8. Both the size of requests and randomness

Table 4
Idle power consumption of each device.

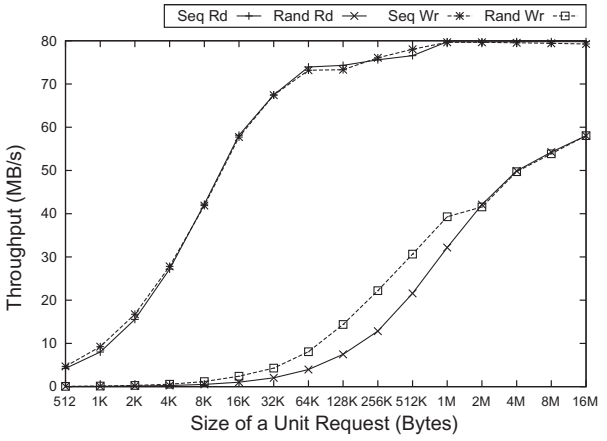
Device	Power (W)	Device	Power (W)	Device	Power (W)
OLD	1.075	MLC	0.519	HDD	0.869
SLC	0.937	TGN	0.210	–	–



(a) SSD read



(b) SSD write



(c) HDD read/write

Fig. 8. Block-level read/write performance.

are critical factors that affect throughput. The SSDs gave a similar performance for sequential reads and random reads as shown in Fig. 8. This is because flash memory has no positioning overhead.

However, sequential writes and random writes on the SSDs have a big performance gap as shown in Fig. 8. This is because random writes increase the merge overhead when making free space. This means that the random writes generate a larger number of erase and valid copy operations than sequential writes do. The throughput difference between random write and sequential write persists even for the requests of 16 MB size. This implies that changing the offset in consecutive write requests requires a significant temporal overhead. TGN has the biggest difference between random writes and sequential writes. The slow erase and write operations of the MLC flash memory and its FTL mechanism seem to be the causes of this significant difference.

Generally, SSDs have larger flash memory than their marketed capacity. This extra space is set aside for spare free blocks or log blocks to improve write performance by reducing the chance of garbage collection during a burst period. However, after writing over all the free or log blocks, the SSDs cannot avoid garbage collections, which are required for making erasable blocks.

Fig. 9 shows time-series measurements of throughput and power consumption of MLC during a random burst write period. In this experiment, we issue 4 MB random writes continually, until the cumulative size of those requests reach 1 GB. Until the cumulative size of the processed requests reaches approximately 300 MB, the throughput is similar to that of sequential requests of the same size. After that, the throughput splits into two different values. Low throughput value is for the throughput with merge operations and high throughput value is for the normal throughput without merge operations. During the experiment, power consumption is between 50 W and 60 W. However, after handling all requests, power consumption periodically hit the double of its idle power. Based on this observation, we verified that the MLC prepares log blocks of about 300 MB during the idle period and it takes a few minutes.

Because both MLC and TGN showed this behavior, we believe that both of them employ hybrid FTLs using log blocks, while the others use the page-mapping FTL or its variants.

Fig. 10 shows the power consumption of SSD reads. SSDs consume similar power for both random and sequential read operations. Power consumption grows as the throughput grows. The

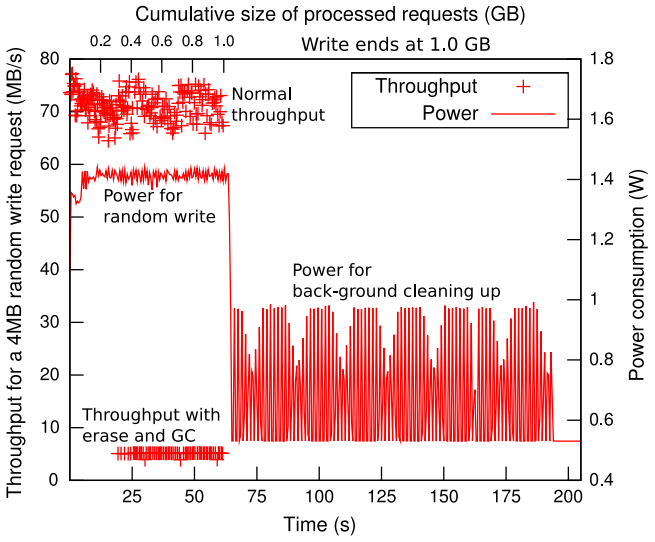
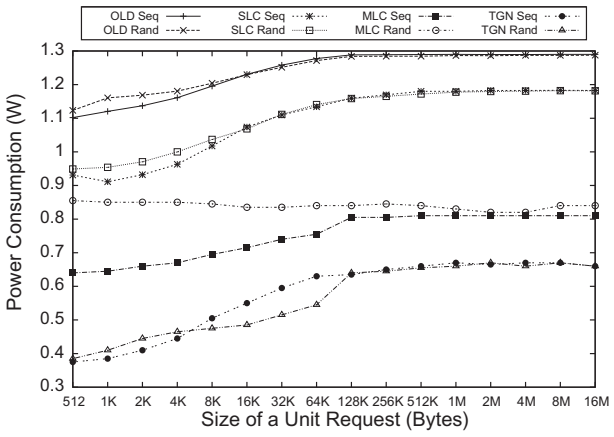
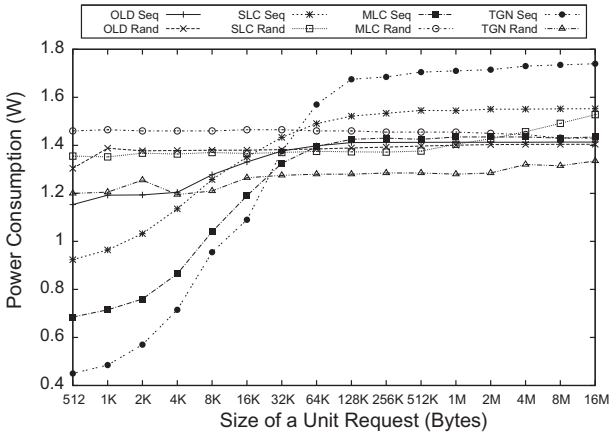


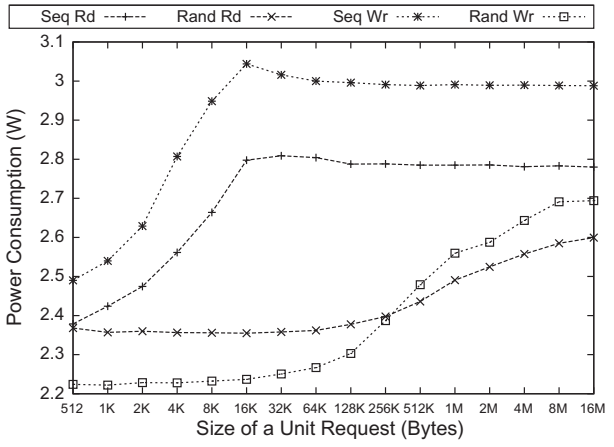
Fig. 9. Throughput and power consumption of MLC for a burst period of continual 4 MB random writes that finally make up to 1 GB write.



(a) SSD read



(b) SSD write



(c) HDD read/write

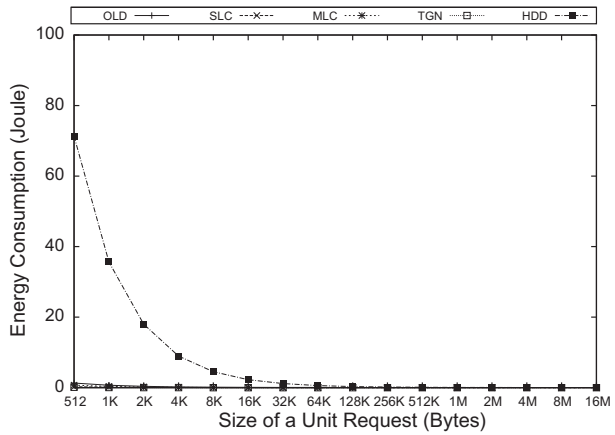
Fig. 10. Power consumption of block-level read/write requests depending on different request sizes.

amount of increased power is due to the increased transferring operation.

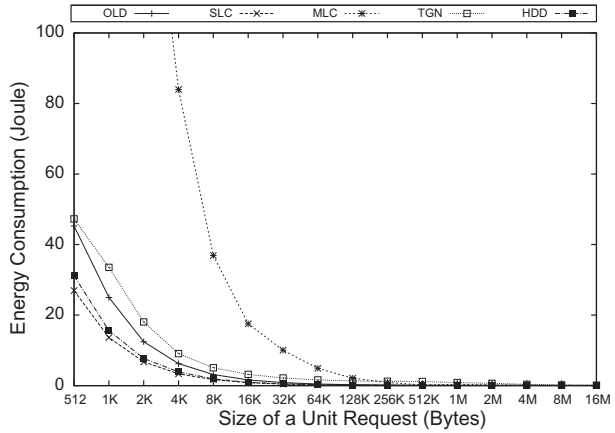
On the contrary, power consumption for writes on SSDs shows a dramatic difference between random and sequential patterns as shown in Fig. 10. Random writes consume almost the highest power regardless of the request size. Only OLD consumes more power when sequentially writing over 1 MB data. This is because the power consumption for transferring data starts to exceed the power consumption for processing the randomness.

Both the high power consumption and the low throughput of random write induce low energy efficiency. We measure the energy efficiency for each case using the same amount of data. As shown in Fig. 11, the required energy of random writes for OLD and MLC is higher than HDD. SLC and TGN also do not show a clear superiority over HDD.

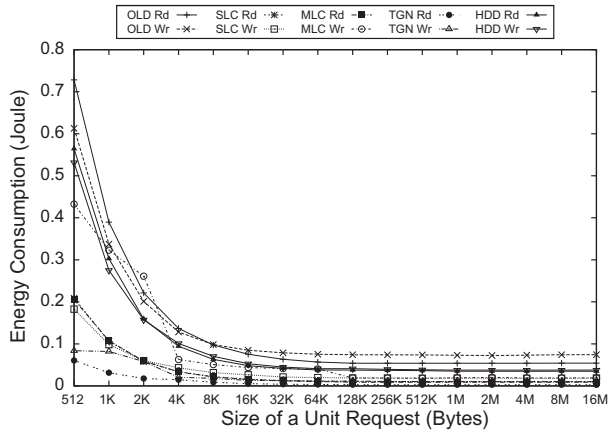
In the case of HDD, random writes with a small request size have better energy efficiency than random reads because the read-ahead for small random reads spends more energy without any performance gain, and with write buffering and reordering, head movement is dramatically reduced. The random reads of the SSDs show substantially better energy efficiency than the HDDs in Fig. 11. They present similar energy values with sequential reads of SSDs. The energy efficiency of sequential access is similarly good with all the devices, presenting under 1 J/MB as shown in Fig. 11c.



(a) Read 1MB randomly



(b) Write 1MB randomly



(c) Read/write 1MB sequentially

Fig. 11. Energy consumption to process 1 MB of data with different request sizes.

In spite of low energy efficiency of small random writes on SSDs, the actual power consumption value of them is not higher than that of the laptop hard disk. The primary reason of the low energy efficiency is extremely slow performance. Therefore, the low energy efficiency does not cause any trouble in high density storage systems or large scale RAID systems. However, if they process intensive small random writes in energy-limited environments like laptops or smart phones, they will reduce the battery lifetime significantly.

For example, a six-cell battery, which is currently used in laptops, contains 50 Wh of energy. When MLC is used to write data on random locations at 4 KB granularity, the battery will be used up after writing 2.2 GB of data. However, if we use HDD instead of MLC, the battery will last until it writes 46 GB of data. Also, if the random writes are transformed into sequential writes, the amount of data to be written to MLC and HDD with the battery will be 3 TB and 1.8 TB, respectively. As a result, we expect that using SSDs for workloads with massive small random writes such as peer-to-peer file sharing or web file caching creates little benefit in terms of energy consumption as well as performance.

These results strongly support the assertion from Narayanan et al. [25] that using SSDs instead of low power laptop hard disks does not show a noticeable benefit from the viewpoint of the ratio between energy efficiency and price to own. While they simply compared the energy consumption and performance of SSDs to that of HDDs under general filesystem level workloads, we analyzed the characteristics under diverse request patterns at the block device level. Our analysis reveals that the reasons behind the claim are the extremely low performance and comparable power consumption of SSDs for small random writes.

While an intuitive way to improve the performance of a hard disk is increasing the RPM of its platters, currently the most popular way to improve the performance of a flash-based SSD is increasing the parallelism of access to its flash memory. Considering the results of Fig. 10 and Table 4, we can tell that increased performance through SSDs' channel parallelism is not a dominant factor for determining power consumption. This is contrary to hard disks, in which increased RPM increases power consumption as well as performance [13].

4.3. Macrobenchmark

The request patterns of a workload at the block device level depends on the filesystem that the storage device is using. When a file is fragmented into many pieces and scattered over entire disk space, sequential read over that file will create multiple random read requests to the file fragments. On the contrary, random writes over multiple files will be transformed into sequential writes at the block device level, if the overlying filesystem allocates write space sequentially on the disk.

In this paper, in order to observe the energy efficiency and performance changes depending on underlying filesystems, we experiment two filesystems with different characteristics, ext2 and LFS.

ext2 is one of the quintessential UNIX filesystems, including UFS, NTFS, ext3 and ext4, which manage files through index node structures. They accommodate data blocks belonging to a file on closely located free blocks so that the seek distance of the hard disk heads is minimized. Also, because hard disks allow in-place update operations, the locations of files are the same after they are updated. In other words, they reduce the performance drawback from seeking operations by exploiting the in-place update ability of hard disks.

Those conventional filesystems have worked well for hard disks. However, considering that random reads on SSDs performs at the same speed as sequential reads and SSDs are unable to conduct an in-place update, it is natural to expect that they cannot perform as well on SSDs as they do on hard disks.

LFS allocates space for incoming write requests on contiguous sequential blocks like logging on a sheet of white paper. It also records the locations of file fragments on metadata blocks, which are stored together with data blocks and written sequentially like data blocks. Therefore, all write requests including writes for recording metadata on LFS only advances the disk head sequentially. In order to reclaim free blocks from invalidated garbage blocks, LFS employs a segmented disk space management scheme and garbage collection operation.

We can expect that LFS would have better performance for the workload with many random writes because it transforms random writes into sequential writes. However, sequential reads of files that have been updated many times is transformed into multiple random reads. This is one of the weak points that have impeded the LFS from being used as a general filesystem because the workload on average has a greater number of read requests than write requests.

To study throughput and power consumption differences due to overlying filesystems, we use fileBench [31], which is a benchmark framework to emulate various filesystem workloads. We select two workload models; *varmail* and *fileserver*. The varmail benchmark simulates filesystem workloads of mail servers or news servers. The average size of write requests is 16 KB and the average file size is 128 KB. The read to write ratio is 1:1 and the total file set size is about 1.5 GB. The write requests in varmail workload are small and random. Therefore, this workload is a representative example for which the LFS performs better than the ext2 filesystem. The fileserver benchmark simulates a file server. We modify the original fileserver workload shipped in FileBench to clearly reveal the flaw of LFS. Each file is written sequentially one by one and updated about thirty times by random writes. After that, 50 reader threads simultaneously read them in a sequential manner. The average file size is 8 MB and the average update size is 16 KB. The total file set size is about 8 GB. Performance and power consumption is measured only for the reading stage.

Fig. 12 shows the FileBench results regarding throughput. In the benchmark of the fileserver on both ext2 and LFS, SSDs impress with their read performance, which is significantly faster than that of the hard disks. However, in varmail on ext2, they perform similarly or slightly better than HDD because varmail has numerous random writes, which are the weak point of both SSDs and HDDs. By transforming random writes into sequential writes, using LFS for the varmail benchmark dramatically improves the performance on all the disks used in our evaluation. In particular, LFS boosts the performance of SLC by about 2.5 times.

LFS transforms sequential read operations into multiple random reads, when the target file has been overwritten multiple times. Therefore, using LFS for fileserver, unlike varmail, decreased the performance for HDD and MLC.

Fig. 12 shows the FileBench results in terms of energy. Because fileserver has read-dominant requests, SSDs consume less energy than HDD except with respect to OLD with varmail. The throughput, in addition to the energy efficiency of fileserver with LFS on HDD, is significantly decreased from that with ext2. However, there is no significant drawback caused by using LFS on the SSDs.

Although MLC performs the worst in terms of throughput and energy efficiency of random writes as evidenced by microbenchmarks, its varmail performance on Ext2 is better than that of OLD and HDD. The performance is boosted by using a large amount of spare free blocks, as explained in Fig. 9. With these results, we are assured that preparing a large amount of free blocks during idle time is a better solution for improving random write performance

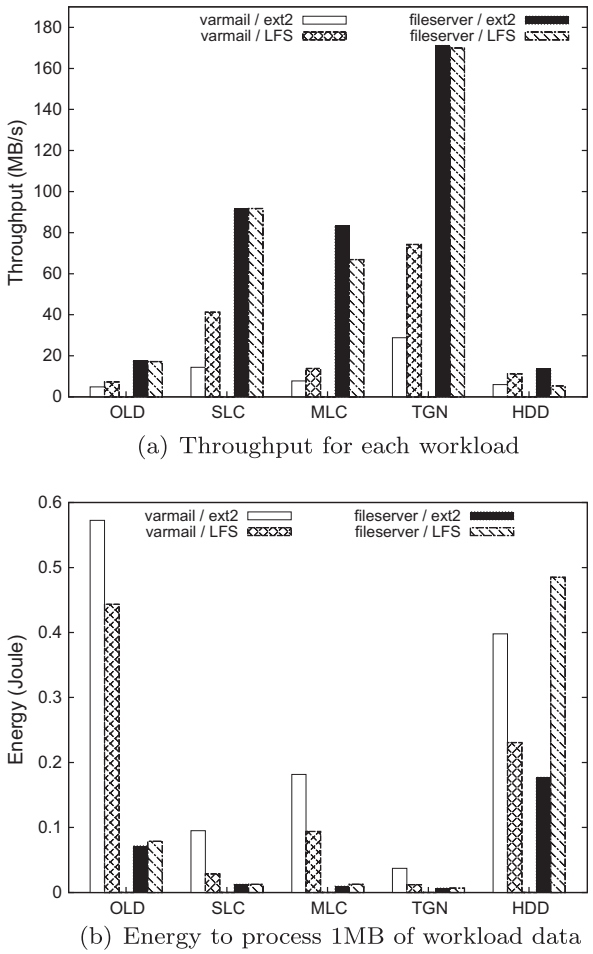


Fig. 12. Throughput and energy efficiency change to the overlying filesystems for *Filebench* workload.

in comparison to using a large amount of write buffer when idle power is important.

As shown in these results, SSDs benefits from employing LFS in terms of energy efficiency and performance, because SSDs are not affected by random read requests transformed from sequential read requests at the file level, which is a critical weak point of LFS. In fact, many modern LFS variants including *nilfs* [20], which is supported by the Linux kernel, are expected to be widely used with SSDs in the near future.

5. Conclusion

For their performance, robustness, and low power consumption, storage systems using flash memory are rapidly becoming popular in diverse computing systems, from embedded systems equipped in consumer electronics to enterprise-scale server systems. Flash-based SSDs are the most common form of the flash-based storage systems. Because they employs FTLs, SSDs can replace the existing hard disks without any modification of software or hardware systems.

Yet, despite its value, the FTL generates significant computational and spatial overhead, which increases power consumption significantly. This paper analyzed the overhead of the three classic FTLs as well as a cutting-edge FTL algorithm in terms of performance and energy efficiency. Also, we empirically analyzed the performance and energy efficiency characteristics of the commodity flash-based SSDs with different hardware configurations at both the block-device level and filesystem level.

We believe that this work will provides valuable insights into devising smarter ways to utilize flash-based SSDs for better energy efficiency and performance than blindly using existing filesystems.

References

[1] Oltp trace from umass trace repository, 2002. Available from: <http://traces.cs.umass.edu/index.php/Storage/Storage>.
[2] Tn-46-03: Calculating memory system power for ddr, Technical report, Micron Technology Inc., 2005.
[3] Solid State Drive MSP-SATA7000 Datasheet, MTRONStorage Technology Co., LTD, 2008.
[4] Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J.D., Manasse, M., Panigrahny, R., 2008. Design tradeoffs for ssd performance, in: Proceedings of USENIX '08 Annual Technical Conference, pp. 57–70.
[5] A. Birrell, M. Isard, C. Thacker, T. Wobber, A design for high-performance flash disks, SIGOPS Operating Systems Review 41 (2) (2007) 88–93.
[6] J.S. Bucy, G.R. Ganger, Contributors, The disksim simulation environment version 3.0 reference manual, Technical Report CMU-CS-03-102, Carnegie Mellon University, 2003.
[7] R. Card, T. Ts'o, S. Tweedie, Design and implementation of the second extended filesystem, in: Proceedings of the First Dutch International Symposium on Linux, 1994.
[8] L.-P. Chang, Hybrid solid-state disks: combining heterogeneous nand flash in large ssds, ASP-DAC '08: Proceedings of the 2008 conference on Asia and South Pacific design automation, IEEE Computer Society Press, Los Alamitos, CA, USA, 2008.
[9] F. Chen, D.A. Koufaty, X. Zhang, Understanding intrinsic characteristics and system implications of flash memory based solid state drives (2009) 181–192.
[10] Corporation, I., Intel Application Note, AP-684, chapter Understanding the Flash Translation Layer (FTL) Specification, Intel Corporation, 1998.
[11] M. Greenberg, How much power will a low-power sdram save you? Denali Software White Paper (2009).
[12] A. Gupta, Y. Kim, B. Urgaonkar, DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings, ASPLOS '09: Proceeding of the 14th International conference on Architectural support for programming languages and operating systems, ACM, New York NY, USA, 2009.
[13] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, H. Franke, Reducing disk power consumption in servers with drpm, IEEE Computer 36 (12) (2003) 59–66.
[14] J. Zhang, A. Sivasubramaniam, H.F.N.G.Y.Z. Nagar, S. Synthesizing representative I/O workloads for tpc-h, in: Proceedings of the International Symposium on High Performance Computer Architecture (HPCA), 2004.
[15] M. Jambor, T. Hruby, J. Taus, K. Krchak, V. Holub, Implementation of a linux log-structured file system with a garbage collector, SIGOPS Oper. Syst. Rev. 41 (1) (2007) 24–32.
[16] A. Kawaguchi, S. Nishioka, H. Motoda, A flash-memory based file system, in: Proceedings of the Winter 1995 USENIX Technical Conference, 1995.
[17] H. Kim, S. Ahn, BPLRU: a buffer management scheme for improving random writes in flash storage, in: Sixth USENIX Conference on File and Storage Technologies, 2008.
[18] H. Kim, E.H. Nam, K.S. Choi, Development platforms for flash memory solid state disks, in: Proceedings of ISORC 2008, 2008, pp. 527–528.
[19] J. Kim, J.M. Kim, S. Noh, S.L. Min, Y. Cho, A space-efficient flash translation layer for compactflash systems, IEEE Trans. Consumer Electron. 48 (2) (2002) 366–375.
[20] J.B. Layton, Nilfs: a file system to make ssds scream, Linux Magazine (2009).
[21] H.G. Lee, N. Chang, Low-energy heterogeneous non-volatile memory systems for mobile systems, J. Low Power Electron. 1 (1) (2005) 52–62.
[22] S. Lee, D. Shin, Y.-J. Kim, J. Kim, Last: locality-aware sector translation for nand flash memory-based storage systems, SIGOPS Oper. Syst. Rev. 42 (6) (2008) 36–42.
[23] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, H.-J. Song, A log buffer-based flash translation layer using fully-associative sector translation, ACM Trans. Embed. Comput. Syst. 6 (3) (2007) 18.
[24] S.-P. Lim, S.-W. Lee, B. Moon, Faster ftl for enterprise-class flash memory ssds, in: Proceedings of the Sixth IEEE International Workshop on Storage Network Architecture and Parallel I/Os, 2010.
[25] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, A. Rowstron, Migrating server storage to ssds: analysis of tradeoffs, in: Proceedings of the 4th ACM European Conference on Computer Systems, 2009, pp. 145–158.
[26] M. Rosenblum, J.K. Ousterhout, The design and implementation of a log-structured file system, ACM Trans. Comput. Syst. 10 (1) (1992) 26–52.
[27] D. Schmidt, N. Wehn, Dram power management and energy consumption: a critical assessment., SBCCI '09: Proceedings of the 22nd Annual Symposium on Integrated Circuits and System Design, ACM, New York, NY, USA, 2009.
[28] E. Seo, S. Maeng, D. Lim, J. Lee, Exploiting temporal locality for energy efficient memory management, Journal of Circuits, Systems, and Computers 17 (5) (2008) 929–941.
[29] T. Shinohara, Flash memory card with block memory address arrangement, United States Patent, No. 5,905,993.
[30] Transaction Processing Performance Council, Tpc-h, 2009. Available from: <http://www.tpc.org/tpch>.
[31] A. Wilson, The new and improved FileBench, in: Proceedings of Sixth USENIX Conference on File and Storage Technologies, 2008.
[32] J.H. Yoon, E.H. Nam, Y.J. Seong, H. Kim, B. Kim, S.L. Min, Y. Cho, Chameleon: a high performance flash/FRAM hybrid solid state disk architecture, IEEE Comput. Archit. Lett. 7 (2008) 17–20.

Seonyeong Park received the B.S. degree in computer science from Chungnam National University and the M.S. degree in computer science from Korea Advanced Institute of Science and Technology in 2001. From 2001 to 2003, she was a researcher at ETRI. Currently she is a Ph.D. candidate in the Computer Science Division at KAIST. Her research interests include flash memory filesystems, embedded systems and power-aware computing.

Youngjae Kim received the B.S. degree from Sogang University in 2001 and the M.S. degree from the Korea Advanced Institute of Science and Technology in 2003. He earned his Ph.D. degree from the Pennsylvania State University in 2009. From 2003 to 2004, he was a research at ETRI Korea. Currently he is a researcher at Oak Ridge National Lab. His research interests include operating systems and power and thermal management of storage systems.

Bhuvan Urgaonkar received his B. Tech in computer science and engineering at the Indian Institute of Technology Kharagpur in 1999, M.S. in computer science at the University of Massachusetts in 2002, and his Ph.D. in computer science at the University of Massachusetts in 2005. He is an assistant professor in the Department of Computer Science and Engineering at the Pennsylvania State University. His research interests include operating systems, storage systems, virtual machines, performance evaluation, and distributed computing.

Joonwon Lee received the B.S. degree in computer science from Seoul National University in 1983 and the M.S. and Ph.D. degrees from the Georgia Institute of Technology in 1990 and 1991, respectively. Until 2008, he had been a professor at Korea Advanced Institute of Science and Technology. Currently, he is a professor at Sungkyunkwan University. His current research interests include low power embedded systems, system software, and virtual machines.

Euiseong Seo received his B.S., M.S., and Ph.D. degree in computer science from Korea Advanced Institute of Science and Technology in 2000, 2002, and 2007, respectively. He is currently an assistant professor at Ulsan National Institute of Science and Technology, Korea. Before joining UNIST in 2009, he had been a research associate in the department of computer science and engineering at the Pennsylvania State University. His research interests are in power-aware computing, real-time systems, embedded systems, and virtualization.