PAPER Special Section on Data Engineering

# Performance Evaluation of Flash SSDs in a Transaction Processing System

Yongkun WANG[†a)], Kazuo GODA[††], *Nonmembers,* Miyuki NAKANO[††], *Member, and* Masaru KITSUREGAWA[††], *Fellow*

**SUMMARY** Flash SSDs are being incorporated in many enterprise storage platforms recently and expected to play a notable role for IO-intensive applications. However, the IO characteristics of flash SSDs are very different from those of hard disks. Since existent storage subsystems are designed on the basis of characteristics of hard disks, the IO performance of flash SSDs may not be obtained as expected. This paper provides an evaluation of flash SSDs in transaction processing systems with TPC-C benchmark. We present performance results with various configurations and describe our observations of the IO behaviors at different levels along the IO path, which helps to understand the performance of flash-based transaction processing systems and provides certain references to build flash-based systems for IO-intensive applications.

***key words:*** *Flash SSD; Database; Transaction Processing System*

## 1. Introduction

Flash SSDs are likely to be used in enterprise storage platforms for achieving high performance in data-intensive applications. Many researchers have proposed solutions to modify the current applications with consideration of the characteristics of flash SSDs [1][2][3][4][5][6]. These solutions are effective to improve the potential performance of flash SSDs in an experimental system. In current enterprise systems, storage subsystems are usually virtualized behind a storage network, such as Network Attached Storage (NAS) and Storage Area Network (SAN), with complex software stacks. Optimization techniques along such a long IO path are also important for achieving high performance. The possibility of this kind of optimizations has been preliminarily demonstrated in [7] with a log-structured file system.

The stacks of current storage systems have been well tuned to the characteristics of hard disks for decades. With different characteristics such as seek-less accesses and "erase-before-write" design, flash SSDs may not effectively provide their high performance when they are simply placed into the existing system.

In order to better utilize flash SSDs in IO-intensive systems fully, we need to have a comprehensive understanding of the IO behavior along the IO path. The IO path has been designed to hide the seek latency and utilize the sequential bandwidth. The whole system has been studied and adjusted to provide a good performance on thousands of hard disks. Similar studies based on characteristics of flash SSDs are also strongly required to achieve expected performance improvement. However there are few reports for evaluating detailed behaviors of SSDs.

This paper provides evaluation on the IO behaviors of SSDs running an actual database application. The online transaction processing is taken for the examination. A traditional file system, ext2fs, and a typical log-structured file system, nilfs2, were adopted. We ran TPC-C benchmark with two DBMSs (commercial and open-source), three high-end flash SSDs (Mtron, Intel and OCZ), and two IO schedulers (Anticipatory and Noop). The differences of basic performance of three SSDs have been measured and analyzed. Then, the detailed behavior of IO path has been investigated. Lastly, we have a comprehensive knowledge that design of log-structured file system should be carefully considered for SSDs.

The rest of this paper is organized as follow: Section 2 gives a brief introduction to flash SSD and an experimental study on the basic performance. Section 3 provides the evaluation that we conducted TPC-C benchmark with two different DBMSs and three different SSDs. Section 4 will discuss some SSD-specific features. The related works are summarized in Section 5. Finally, our conclusion and future works are provided in Section 6.

## 2. Flash SSDs and Basic Performance

### 2.1 Flash SSDs

NAND flash memory is a kind of EEPROM (Electrically Erasable Programmable Read-Only Memory). The memory space can be divided into many blocks, each block being composed of multiple pages. There are three operations for NAND flash memory: read, program, and erase. The read and program operations are conducted on an arbitrary page. Once the page is programmed, it cannot be re-programmed directly. An erase operation is required to reset the state of the whole block before the concerned page can be programmed with new data. Read and program operations can complete in tens of microseconds to hundreds, while the erase operations are slow, often taking over a millisecond. Table 1 shows the performance parameters of a typical Samsung 4GB flash memory chip [8]. This "erase-before-write" design leads to the relatively poor performance for random

Manuscript received June 4, 2010.
[†]The author is with Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, 153-8505 Japan
[††]The author is with Institute of Industrial Science, The University of Tokyo, Tokyo, 153-8505 Japan
a) E-mail: yongkun@tkl.iis.u-tokyo.ac.jp
DOI: 10.1587/transinf.E94.D.1

**Table 2** Storage devices in our experimental system, with the performance specifications

| Manufacture Model | SLC/MLC /RPM | Form Factor | Interface | Capacity | Cache Size | Heads/ Channels | Sustained Rate | Performance Value |
|---|---|---|---|---|---|---|---|---|
| HGST HDS72107 [9] | 7200 RPM | 3.5″ | SATA 3.0Gbps | 750GB | 32MB | 8 heads | 300MB/s[1] | Seek time: 8.2ms read (typical) 9.2ms write (typical) |
| Mtron PRO 7500 [10] | SLC | 3.5″ | SATA 3.0Gbps | 32GB | 16MB[2] | 4 channels[3] | Read: 130MB/s Write: 120MB/s | Sequential Read IOPS(4KB): 12,000 Sequential Write IOPS(4KB): 21,000 Random Read IOPS(4KB): 12,000 Random Write IOPS(4KB): 130 |
| Intel X25-E[11] | SLC | 2.5″ | SATA 3.0Gbps | 64GB | 16MB[4] | 10 channels [12] | Read: 250MB/s Write: 170 MB/s | Random Read IOPS(4KB): 35,000 Random Write IOPS(4KB): 3,300 |
| OCZ VERTEX EX [13] | SLC | 2.5″ | SATA 3.0Gbps | 120GB | 64MB | Not found | Read: 260MB/s Write: 100 MB/s | Seek Time: less than 0.1ms |

[1] This bandwidth is connection bandwidth. Sustained transfer rate is not disclosed in the data sheet.
[2] Reported by hdparm[14] in our test system.
[3] Estimated by the number of Flash Bus Controller (FBC) in the block diagram.
[4] Obtained by the memory chip used in the 32GB model.[15]

write.

**Table 1** Parameters of Samsung 4GB flash memory chip

| | |
|---|---|
| Page Read (4KB) | $25\mu s$ |
| Page Program (4KB) | $200\mu s$ |
| Block Erase (256KB) | $1500\mu s$ |

With the rapid development of NAND flash memory chips, the flash SSD (Solid State Drive) has appeared in the markets. The flash SSD is assembled with large-capacity flash memory chips and a dedicated control system. There may be a number of flash channels connected in parallel between the flash chips and the control system. The control system contains the mapping logic called Flash Translation Layer (FTL) and the buffer cache, which together provide disk device emulation. Application running on the server can use traditional block read/write commands to access the data stored in the flash SSD as if the data is stored in the conventional hard disk.

## 2.2 Basic Performance Study

### 2.2.1 System Setup

The flash SSD has the capability of disk emulation, but its internal mechanism is different of that of conventional hard disk. The performance characteristics should be carefully studied when we think about the deployment into data-intensive systems. In this section, we present our experimental examination with three major SSD products.

We built a Linux server on a DELL Precision 390 workstation, with Dual-core Intel 1.86GHz CPU and 2GB memory. Table 2 describes three high-end flash SSDs used in our system from the major product lines of Mtron, Intel and OCZ.† We employed default settings in all the experiments: read-ahead prefetching and write-back caching were enabled. For comparison, HGST's hard disk was also studied in the experiment.

We developed a micro benchmark tool for measuring the IO performance by issuing several types of IO sequences (e.g. purely sequential reads or 50% random reads plus 50% random writes) to a target device. The benchmark tool was set to bypass the file system and the OS buffer in order to clarify the pure performance of the concerned device.

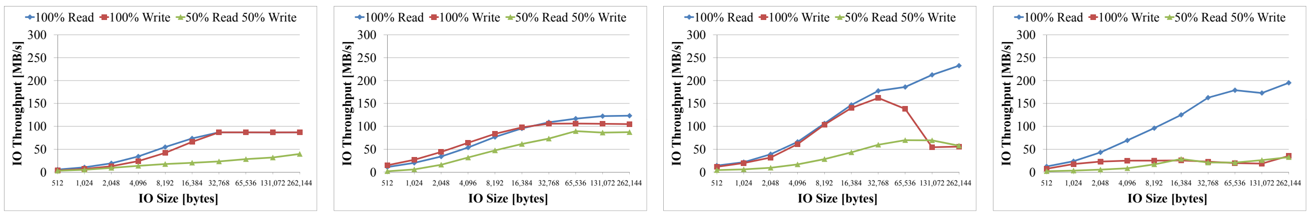
### 2.2.2 IO Throughput

We firstly examined the IO throughput of sequential accesses. Three cases are compared for each device in Fig. 1. Higher throughputs were confirmed in most of the cases in the SSDs than the hard disk, but several exceptions were also seen. In Fig. 1(b), the read and write throughputs of Mtron's SSD were close to each other and saturated at around 120MB/s, which is consistent with the bandwidth specifications in Table 2. In Fig. 1(c), the read and write throughputs of Intel's SSD were much higher than that of Mtron's SSD, but the write throughput decreased when request size became larger than 32768 bytes. The acclaimed bandwidth in Table 2 was also confirmed in Fig. 1(c) when the request size was set to 1MB††. In Fig. 1(d), the read throughput of OCZ's SSD was higher too, but the write throughput was the worst. The confirmed bandwidth here was lower than that indicated in the Table 2.

Random accesses were next studied. Fig. 2(a) to 2(d) show the throughputs that were observed for random accesses when the number of outstanding IOs was set to one.

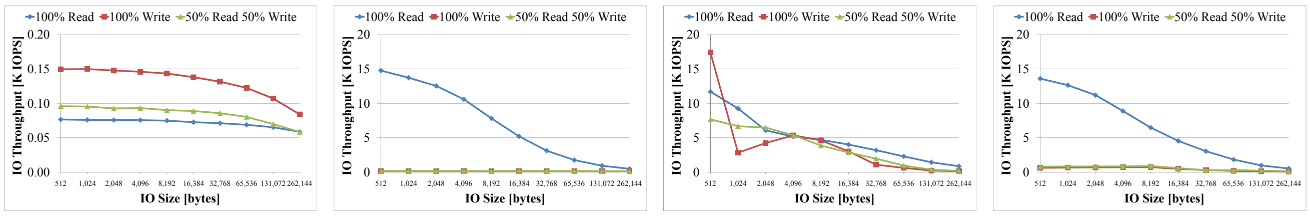
†Table 2 summarizes internal design information such as cache size and channel number that vendors have disclosed. As is mentioned later in section 2.2.2, this limited information is not enough to explain overall performance of SSDs. The overall performance is affected by complex of various design factors such as flash chips' performance characteristics, internal channel latency/bandwidth, controller processing power, external (device to host) interconnection latency/bandwidth and even software algorithm employed in the controller. We could know the basic performance first after we did the experiments described in section 2.2.

††We do not show the data with the request size larger than 262,144(256K) bytes in Fig. 1, 2 or 3 for the brevity.



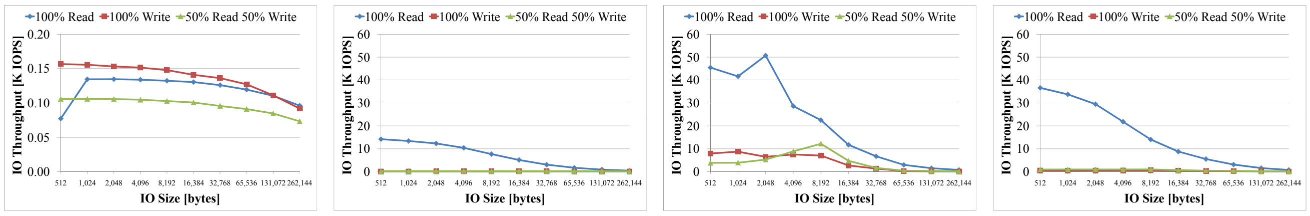
(a) HGST (b) Mtron (c) Intel (d) OCZ

**Fig. 1** IO Throughput for Sequential Access



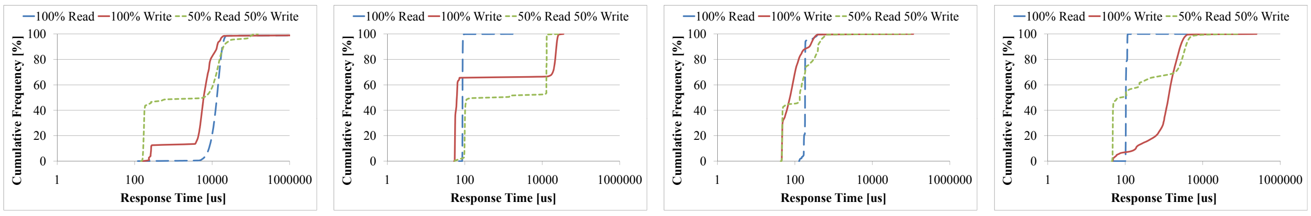
(a) HGST (b) Mtron (c) Intel (d) OCZ

**Fig. 2** IO Throughput for Random Access (Single Thread)



(a) HGST (b) Mtron (c) Intel (d) OCZ

**Fig. 3** IO Throughput for Random Access (Thirty Threads)



(a) HGST (b) Mtron (c) Intel (d) OCZ

**Fig. 4** IO Response Time Distribution for Random Access (Single Thread)

The read throughput kept untouched as that of sequential throughput and much higher than that of the hard disk, while the write and the mixed-access throughputs were drastically degraded on Mtron's SSD and OCZ's SSD. In the case of mixed access, reads and writes were respectively given 50% probability. We had expected that the mixed-access throughput would fall between the read throughput and the write throughput, but the observation was that the mixed-access throughput was comparable with the write throughput. Similar observations are also confirmed by other researchers and this characteristic is sometimes called *bathtub effect*[16]. Only on Intel's SSD, the write and mixed-access throughputs were clearly better than that on hard disk.

The same experiment was conducted with 30 outstanding IOs. The results are shown in Fig. 3(a) to 3(d). The read throughput improved clearly on the hard disk, Intel's SSD and OCZ's SSD, while significant improvement was not confirmed for the read throughput in Mtron's SSD and the write and mixed-access throughputs in all the SSDs.

Let us look back at the vendor-disclosed design information such as cache size and channel number cited in Table 2. Unfortunately we could not find strong relationship between such design information and the experimental results above presented. The overall performance is also impacted by more other design factors, yet vendors only disclosed limited design information. Fig. 1 shows that Intel's SSD has relatively high transfer rate than Mtron's SSD. This might be brought by design difference of internal channels;

Intel's SSD holds ten channels whereas Mtron's SSD has only four. But, even bandwidth of internal channels, namely MB/s, is not disclosed. Further analysis is not possible for us due to the lack of information. Experiment presented in this section can be seen as an alternative way for us to understand the basic performance characteristics.

#### 2.2.3 IO Response Time

Since the random access performance is important to typical database applications such as the transaction processing systems, we further studied the response time. The response time distribution is shown in Fig. 4. We can have the following observations:

*Random Read* The random read response time was close to each other among three SSDs. Note that we saw a single sharp cliff in each cumulative frequency curve for the SSDs. This means that most of random reads could complete in a very small range of response times. Such small variance in response times was not confirmed in conventional hard disk, where the rotational platter always gives unpredictability of response times.

*Random Write* Compared with random read, random write gave more complicated characteristics. Two major cliffs were confirmed around 100 microsecond and 10,000 microsecond respectively in Mtron's SSD. Although the inside logic is not documented, our conjecture is that the long response time is caused by the inside flush operations. When the on-disk buffer is full, the control system will flush some pages and make room for the new requests. The flush operation is very time-consuming since it may incur the erase operations. Similarly, multiple cliffs were also confirmed in Intel's SSD and OCZ's SSD.

*Random 50% Read 50% Write* This pattern is close to *Random Write*. Although the pure read performance was very high and almost predictable, the write performance and the mixed-access performance were sometimes much poor and its variance was significant.

## 3. Performance Evaluation By TPC-C Benchmark

We present our experimental examination of three major SSDs with a standard benchmark TPC-C.

### 3.1 Experimental setup

We built a database server on the same system described in Section 2.2.1. The software stacks can be illustrated in Fig. 5. We will describe it in a top-down manner.

In Fig. 5, TPC-C [17] benchmark is at the top of our experimental system. TPC-C is a standard benchmark simulating real online transaction processing workloads. It is actually accepted by many hardware and software vendors. The workload of TPC-C is composed of two read-only and three update transactions, which together provide many random reads and writes to the storage device.

In our TPC-C benchmark application, we started 30 threads to simulate 30 virtual users with 30 warehouses. The initial database size was 2.7GB. The *Key and Thinking* time was set to zero in order to measure the maximum performance. The mix of the transaction types is shown in the *normal* column of Table 3. Unless specially stated, we used this mix for the experiment. Besides the normal mix in Table 3, we also configured another two types of workloads: *read intensive* and *write intensive*, in which the *read-only* and *read-write* transactions are dominant respectively.

DBMS serves the requests from TPC-C benchmark. In the experiment, we set up a commercial DBMS and an open-source DBMS MySQL. The detailed configuration of these DBMSs is shown in Table 4.

We selected two file systems for evaluation, the traditional EXT2 file system (ext2fs) and a recent implementation of log-structured file system, nilfs2 [18]. The block size was set to 4KB for both of them. The garbage collection (GC) is disabled by default in nilfs2 for the simplicity of analysis. We will also show the influence of GC in Section 3.2.3.

We also used two IO schedulers, Anticipatory and Noop in this Linux server. By default, the Anticipatory was used in the experiment because it is the default one in our Linux distribution.

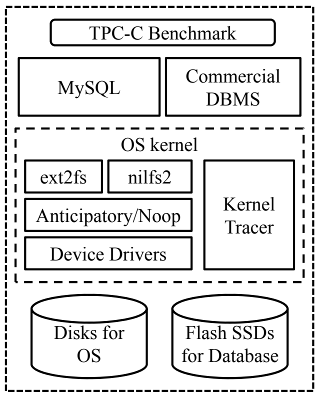


**Fig. 5** Stack of system configuration

**Table 3** Transaction types in TPC-C benchmark

| Transaction Type | IO Property | % of mix | | |
|---|---|---|---|---|
| | | normal | read intensive | write intensive |
| New-Order | read-write | 43.48 | 4.35 | 96.00 |
| Payment | read-write | 43.48 | 4.35 | 1.00 |
| Delivery | read-write | 4.35 | 4.35 | 1.00 |
| Stock-Level | read-only | 4.35 | 43.48 | 1.00 |
| Order-Status | read-only | 4.35 | 43.48 | 1.00 |

### 3.2 Transaction Throughput

#### 3.2.1 Transaction Throughput

Fig. 6 shows the transaction throughput in terms of

**Table 4** Configuration of DBMS

| | Commercial DBMS | MySQL(InnoDB) |
|---|---|---|
| Data buffer size | 8MB | 4MB |
| Log buffer size | 5MB | 2MB |
| Data block size | 4KB | 16KB |
| Data file | fixed, 5.5GB, database size is 2.7GB | |
| Synchronous IO | Yes | Yes |
| Log flushing method | flushing log at transaction commit | |

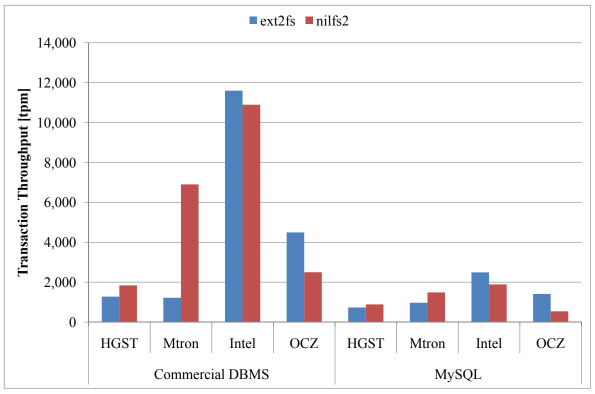


**Fig. 6** Transaction Throughput

transactions-per-minute (tpm).

The advantage of flash SSDs over hard disk is clear; the transaction throughput on the SSDs was higher than that on HGST in either DBMS.

For Mtron SSD, a noticeable improvement of nilfs2 over ext2fs on commercial DBMS was observed. This stemmed from the log-structured design that nilfs2 holds. That is, in nilfs2, every time a write is requested, the file system allocates a new space for that request. This helps to avoid the time-consuming erase operation on flash SSDs. Even if DBMS requests a sequence of random writes to the file system, it can give a converted sequence of virtually sequential writes. See again Section 2.2, where we confirmed that Mtron's SSD has higher throughput of sequential writes rather than random writes. Nilfs2 successfully exploited this characteristic to derive improved performance. However, contrary to expectation, the advantage of log-structured file system is not clear in Intel's and OCZ's SSDs. We gives analysis on this point in later sections.

### 3.2.2 IO Throughput

So as to understand the system behavior more, we traced in-kernel IO events by using SystemTap[19]. Fig. 7 shows the throughput of file system access given by DBMS under the TPC-C execution. For reference, let us call these file system accesses *logical IOs*. The workload nature of TPC-C is IO intensive. The overall transaction throughput is mainly determined by the available IO power. Seeing Fig. 6 and Fig. 7, we could verified that the transaction throughputs actually followed the logical IO throughputs. Note that the logical IOs may not directly go to the storage device, but rather be split, merged or buffered by the file system. Further analysis is required on the IOs in the underlying layers.

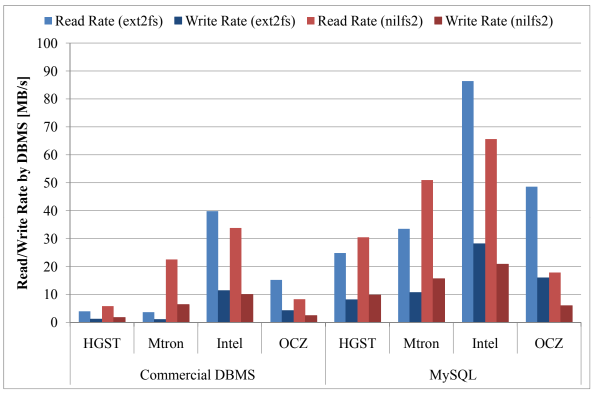


**Fig. 7** Logical IO Rate

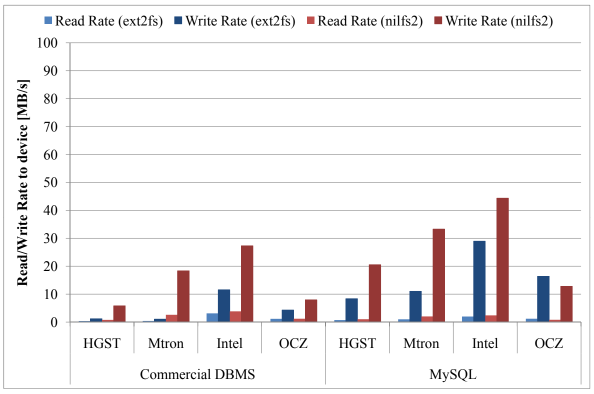


**Fig. 8** Physical IO Rate

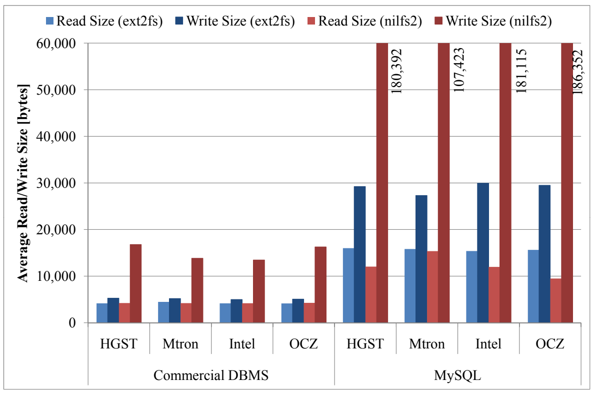


**Fig. 9** Average IO Size

In order to understand the IO path thoroughly, we also analyzed how these logical IOs are processed in the underlying layers. Fig. 8 presents the throughputs of storage device accesses in the same execution. Let us call these accesses *physical IOs*. The physical IO rate is the consequence fueled by the file system capabilities and the device power. Read throughputs were always higher than write throughputs in Fig. 7, whereas write throughputs were higher in Fig. 8. This means that the file system absorbed many read requests in its buffer.

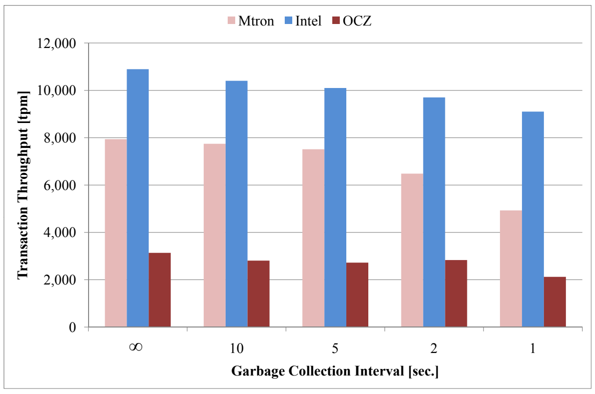
When ext2fs is used, write throughputs are almost the same between logical throughput and physical throughput. It is probably because that write requests are temporarily stored in the file system buffer, but most are directly flushed out to the storage device.

In contrast, when nilfs2 is used, write requests are more eagerly optimized. As is mentioned before, nilfs2 has employed the log-structured design. Each time a write is requested, a new storage block is allocated and the write request is routed to the block. This helps avoiding slow erase operations in the flash SSDs. It is clear in Fig. 9 that the write size of nilfs2 is larger than that of ext2fs because nilfs2 has coalesced the random writes into large sequential writes. Large sequential request is beneficial on hard disks and some SSDs. Actually, we could improve the transaction throughput by using nilfs2 in Mtron's SSD. However, our observation also suggests that we cannot ignore two possible drawbacks of this strategy. First, log-structured strategy has the possibilities of producing more writes. This was confirmed by Fig. 7 and Fig. 8. More writes were issued at the physical layer than the logical layer. Even if nilfs2 can improve the IO throughput by converting random writes into sequential writes, additional writes may finally degrade the overall application performance. Second, too large IO sizes have the possibilities of degrading the throughput. In Intel's and OCZ's SSDs, sequential performance decreases when the request size is larger than 32KB, as discussed in Section 2.2.2. This explains why the physical write rate of nilfs2 on Intel and OCZ's SSD in Fig. 8 is much better than that of ext2fs for the commercial DBMS because the average write size is smaller. For MySQL, since the request size is very large (100KB+), the physical write rate of nilfs2 on Intel's SSD is not so much better than that of ext2fs, on OCZ's SSD it is even worse than that of ext2fs. Note that although the write IO rate of nilfs2 is always higher than that of ext2fs for Intel's SSD, the transaction throughput of nilfs2 is lower than that of ext2fs.

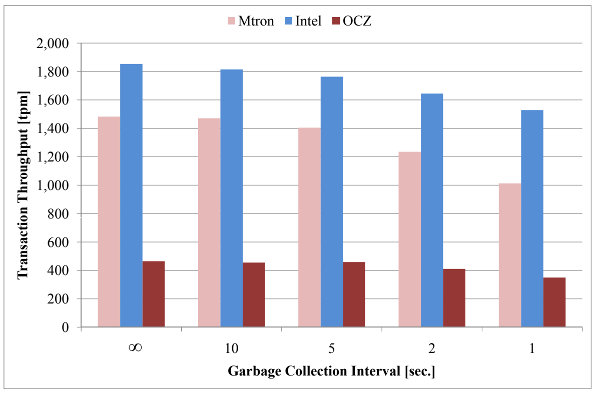
### 3.2.3 Garbage Collection

The log-structured file system tries to allocate a new data block for writing a data, even if it overwrites the existing data. This strategy produces lots of invalid blocks when write-intensive workload runs for a long time. Garbage collection (GC), a.k.a. segment cleaning, is an essential function, which collects such invalid blocks and makes them reusable for future writes.

So far, we have done the experiments with garbage collection disabled. This was intended for us to measure the potential performance of the system. In real systems, peak workloads may not continue so long and disabling garbage collections can be an acceptable solution in such limited time. But, garbage collection is also an inevitable topic when we think about long-time operation. We also studied the influence of garbage collection. We got the transaction throughput with different cleaning interval as shown in Fig. 10. Monotonic performance degradation was observed in all the experimental cases. As garbage collection occurred more frequently, the transaction throughput decreased more. The degradation ratios were varied around 0.77% – 10.44% with a moderate configuration (10 seconds) and 17.51% – 38.83% even with a severe configuration (1 second). Mtron's SSD for both DBMSs and OCZ's SSD for commercial DBMS were relatively sensitive to garbage collection, while Intel's SSD for commercial DBMS and MySQL and OCZ's SSD for MySQL were less sensitive.



(a) Commercial DBMS



(b) MySQL

**Fig. 10** Transaction Throughput with Garbage Collection Enabled

## 3.3 Transaction Throughput by Various Configurations

### 3.3.1 Buffer Size

The database buffer plays a vital role to the cache hit rate, the write merging and re-ordering. The buffer size is influential to the performance. The complexity is that DBMS reserves some portion of the available main memory for the database buffer, but the remaining memory space is also used as the buffer cache for the file system, where some optimizations may be tried too. Fig. 11 shows the transaction throughputs that we measured by varying the buffer size on Mtron's SSD. The absolute performance increased as we increased the buffer size of two DBMSs on both file systems. However, the performance speedup of nilfs2 to ext2fs decreased.

With large database buffer size, a lot of random writes can be cached in the database buffer, the amount of random writes reaching to the file system was greatly reduced. The advantage of log-structured file system is then reduced.

### 3.3.2 Workload Type

In the experiments presented so far, we have only employed a standard mix of transactions. Here we present another two types of workloads, read intensive and write intensive, as indicated in Table 3. As shown in Fig. 12, absolute transaction throughputs were higher for read-intensive workloads. The speedups from ext2fs to nilfs2 were conversely higher for write-intensive workloads.

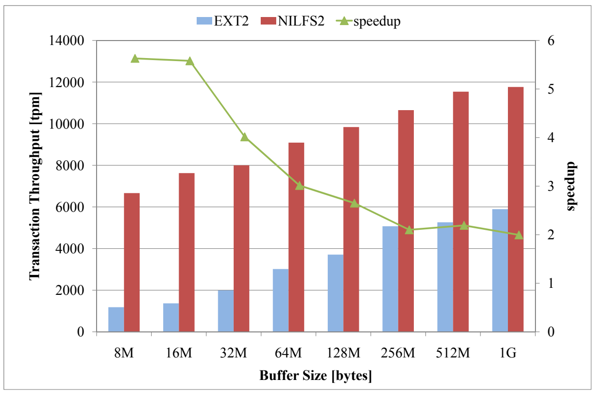
### 3.3.3 IO Scheduler

The IO strategies can also be implemented by different IO schedulers. Two IO schedulers are selected for comparison; Anticipatory and Noop. The Anticipatory scheduler will do the requests merging and ordering, arrange the requests in the one-way elevator and delay some time to anticipate the next request, to reduce the movement of disk head. The Noop scheduler is the simplest one, which only merges the requests, and serves them in FIFO order.

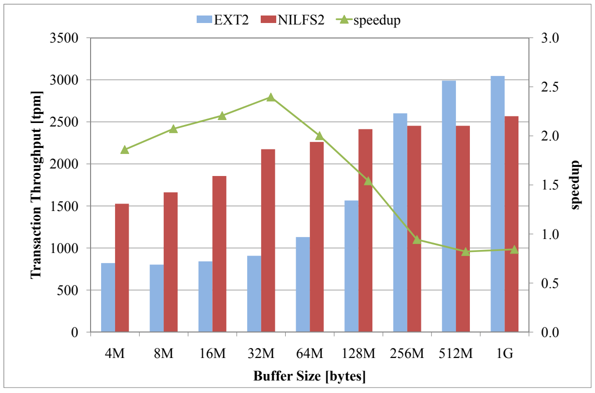
The Noop scheduler is believed to be the best choice for the flash SSDs since there is no mechanical moving parts. Fig. 13 shows the transaction throughput with Noop and Anticipatory schedulers. The Noop scheduler can help to have further improvement in the case of nilfs2 on Mtron SSD and ext2fs on Intel SSD. But it is not clear about superiority over the Anticipatory scheduler in the rest cases, and the Noop scheduler is even worse in some cases.

## 4. Discussion on SSD-Specific Features

One innate characteristic of flash chips is the limited programming cycles, which leads to limited lifespan of SSD. If a particular flash page is written in many times, that page will be worn out (i.e. coming to be unable to hold a written data safely) soon even though other pages are healthy. It would shorten the life time of flash SSDs. Balancing the write count among all the flash pages, often called wear-leveling, is an essential solution. One typical technique is to redistribute hot (frequently written) pages to other places [20][21]. If TPC-C is running on a conventional in-place-update file system such as ext2fs, writes are often skewed on particular pages. This technique seems essential to prolong the life span. When we use a log-structured file system such as nilfs2, the file system itself is self-balancing. That is, it can automatically distribute most of pages over the whole address space in a copy-on-write manner. Potentially wear-leveling could be mostly relieved. But wear-leveling is an internal function that is mainly implemented in SSD controller. Real algorithms are not disclosed by any vendors at present. We would like to study the effect of wear-leveling on the choice of file systems in the future.



(a) Commercial DBMS



(b) MySQL

**Fig. 11** Transaction throughput on Mtron SSD with different buffer size of database system

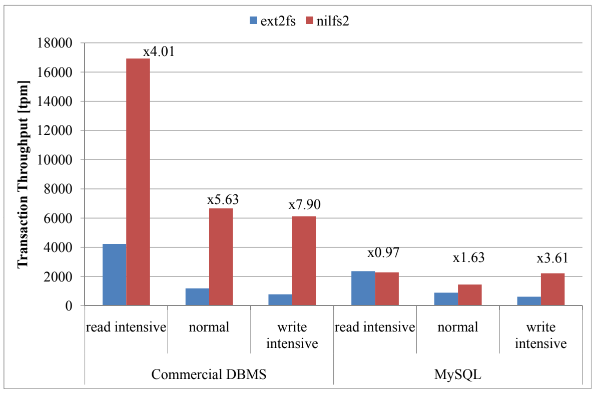


**Fig. 12** Transaction throughput of commercial database with different workload on Mtron SSD

Although the wear-leveling can prolong the lifespan of the whole disk, the overall write operation count is still limited. The limitation of write operation counts is directly related to the reliability of the SSD. We collected the reliability information of each SSD, as shown in Table 5. Given the information in Table 5, we try to roughly calculate the endurance of the SSDs in the transaction processing system by the IO throughput at the driver level in our experiment. Intel discloses in the specification that the SSD we used is guar-

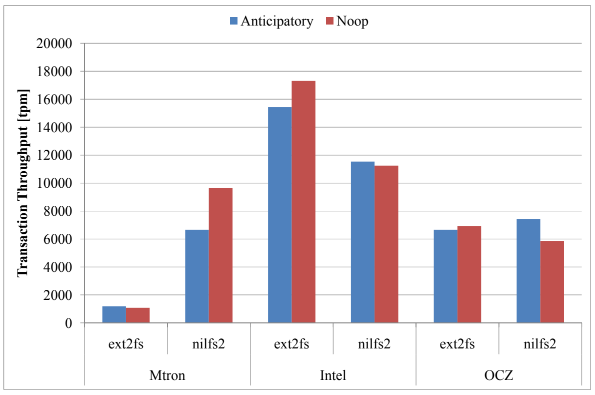


**Fig. 13** Transaction Throughput by different IO schedulers on commercial DBMS

**Table 5** Reliability Information of SSDs

| Manufacture & Model | Reliability Information |
|---|---|
| Mtron PRO 7500 [10] | MTBF[1] : 1,000,000 hours, write endurance is greater than140 years at 50GB write/day at 32GB SSD.[2] |
| Intel X25-E[11] | MTBF[1]: 2,000,000 hours, 64 GB drive supports 2 petabyte of lifetime random writes. |
| OCZ VERTEX EX[13] | MTBF[1]: 1,500,000 hours |

[1] MTBF: Mean Time between Failures.
[2] The above calculation is based on the guaranteed 100,000 program and erase cycles of SLC type flash memory from vendors and the assumption that the write is performed in sequential manner.[10]

anteed two petabytes of lifetime random writes. Random write produces the largest write counts in general. We obtained an expected minimum lifetime by dividing this guaranteed lifetime write amount (in bytes) by average throughput (MB/s) shown in Fig. 8. Mtron also discloses its guaranteed lifetime write amount, but it is measured only for sequential writes. OCZ does not disclose any guaranteed lifetime write amount. We could not obtain an expected lifetime for Mtron's SSD or OCZ's SSD. The result is shown in Table 6. We can see that Intel's 64GB SSD could only last 1.43 years in the shortest case. Note that, in our experiments, TPC-C ran at top speed, namely without any keying or thinking time, in order to measure the potential performance of SSDs for transaction processing. In real systems, most of SSDs may be running at moderate workloads in most of time, and thus they possibly can survive much longer time. Further investigation is necessary on this point. Boboila *et al.*[20] had shown that the endurance of tested flash chips is far longer than the nominal values by manufactures. More solutions such as the redundancy of SSD or fat provision of flash chips could be also considered to improve the reliability.

Another feature specific to SSDs is the TRIM command [22]. When trying to delete a page in a file volume and/or a database, many file systems and/or database systems do not physically erase content of the concerned page, but merely drop a pointer to the page in the volume meta

**Table 6** SSDs endurance in years by the IO throughput in Fig. 8

| | Intel (years) |
|---|---|
| Commercial DBMS with ext2fs | 5.47 |
| MySQL with ext2fs | 2.19 |
| Commercial DBMS with nilfs2 | 2.32 |
| MySQL with nilfs2 | 1.43 |

data (such as inode, directory or catalog). This logical deletion strategy is beneficial in terms of performance, but it would abandon a chance of SSDs to know what page has been deleted by the file systems or the database systems. TRIM, a new storage command, has been proposed as a solution to this. It can inform flash SSDs of which page has been logically deleted, so that the notified SSDs can preemptively erase and release the concerned page. This often helps the performance improvement by hiding slow erase operations. Unfortunately this new command has not been supported in our experimental system, so we could not experiment this. We would like to investigate the effect in the future work.

## 5. Related Work

### 5.1 Log-structured File System

The Log-structured file system (LFS)[23] is designed to exploit fast sequential write performance of hard disk. It can convert the random writes into sequential writes. However the side effect is that the sequential reads may also be scattered into random reads. Overall, the performance can be improved to write-intensive applications. The LFS is also expected to improve the random write performance of flash memory, since the fast read performance of flash memory well mitigates the side effect. For the garbage collection of LFS, an adaptive method based on usage patterns is proposed in [24].

### 5.2 Flash-based Technologies

#### 5.2.1 Flash Translation Layer

FTL bridges the operating system and flash memory. The main function of FTL is mapping the logical blocks to the physical flash data units, emulating flash memory to be a block device like hard disk. Early FTL used a simple page-to-page mapping[25] with a log-structured architecture[23]. It required a lot of space to store the mapping table. The block mapping scheme was proposed in order to reduce the space for mapping table. The scheme introduced the block mapping table with page offset to map the logical pages to flash pages[26]. However, the block-copy may happen frequently. To solve this problem, Kim improved the block mapping scheme to the hybrid scheme by using a log block mapping table[27].

### 5.2.2 File System

Most of the file systems for flash memory exploit the design of Log-structured file system[23] to overcome the write latency caused by the slow erase operations. JFFS2[28] is a journaling file system for flash with wear-leveling. YAFFS[29] is a flash file system for embedded devices. DFS[1] provides a file system design on flash storage layer instead of the FTL layer. The DFS is designed to bypass the traditional file system buffer and perform direct access to the SSDs via the flash storage layer.

### 5.2.3 Database System

Early design for database system on flash memory mainly focused on the embedded systems. FlashDB[6] is a self-tuning database system optimized for sensor networks, with two modes; disk mode for infrequent write and log mode for frequent write. LGeDBMS[30], is a relational database system for mobile phone. For enterprise database design on flash memory, In-Page Logging[5] is proposed. The key idea is to co-locate a data page and its log records in the same physical location.

As flash SSDs are being used in enterprise storage platforms, many researchers are focusing on the performance of the flash SSDs instead of the raw flash memory. Agrawal et al.[31] studied the internal design tradeoffs that will have impact on the performance. Myers[32] had a study on the usage of flash SSD in database systems and provided the IO throughput comparison between the LFS and the conventional file system. We also had evaluation of LFS on SSD using TPC-C benchmark[7][33].

## 6. Conclusion and Future Work

We have presented experimental performance evaluations of three major flash SSDs. First, we have clarified the basic performance characteristics of flash SSDs and compared them with a conventional hard disk. Next, we have measured and analyzed the application performance and the in-kernel IO behavior on three flash SSDs and two file systems with TPC-C benchmark. Finally, we have also studied a variety of configurations for TPC-C. These measurements have provided some practical experiences for building flash-based database systems. The performance benefits of log-structured design were confirmed only in limited cases. It was against our early expectation. The necessity of careful design was verified. We plan to study eager optimization techniques for database applications running on flash SSDs.

**References**

[1] W.K. Josephson, L.A. Bongo, D. Flynn, and K. Li, "DFS: A File System for Virtualized Flash Storage," FAST, 2010.

[2] S. Chen, "Flashlogging: exploiting flash devices for synchronous logging performance," SIGMOD, pp.73–86, 2009.

[3] D. Tsirogiannis, S. Harizopoulos, M.A. Shah, J.L. Wiener, and G. Graefe, "Query processing techniques for solid state drives," SIGMOD, pp.59–72, 2009.

[4] D. Agrawal, D. Ganesan, R.K. Sitaraman, Y. Diao, and S. Singh, "Lazy-adaptive tree: An optimized index structure for flash devices," PVLDB, vol.2, no.1, pp.361–372, 2009.

[5] S.W. Lee and B. Moon, "Design of flash-based DBMS: an in-page logging approach," SIGMOD, pp.55–66, 2007.

[6] S. Nath and A. Kansal, "FlashDB: dynamic self-tuning database for NAND flash," IPSN, pp.410–419, 2007.

[7] Y. Wang, K. Goda, and M. Kitsuregawa, "Evaluating non-in-place update techniques for flash-based transaction processing systems," DEXA, pp.777–791, 2009.

[8] Samsung Corporation, K9XXG08XXM Flash Memory Specification, 2007.

[9] Hitachi Global Storage Technologies, "Deskstar 7K1000 Specifications." `http://www.hitachigst.com/deskstar-7k1000`.

[10] Mtron, "Solid State Drive MSP-SATA7535 Product Specification, revision 0.3." `http://rocketdisk.com/Local/Files/Product-PdfDataSheet-35_MSP-SATA7535.pdf`, 2008.

[11] Intel, "Intel X25-E SATA Solid State Drive, Product Manual." `http://download.intel.com/design/flash/nand/extreme/319984.pdf`.

[12] Intel, "IntelR X25-E Extreme SATA Solid-State Drive, Technical specifications." `http://www.intel.com/design/flash/nand/extreme/index.htm`.

[13] OCZ, "OCZ Vertex EX Series SATA II 2.5″ SSD Specifications." `http://www.ocztechnology.com/products/solid_state_drives/ocz_vertex_ex_series_sata_ii_2_5-ssd`.

[14] hdparm. `http://hdparm.sourceforge.net/`.

[15] "Review: Intel X25-E 32GB SSD." `http://www.bit-tech.net/hardware/storage/2008/12/17/intel-x25-e-32gb-ssd-review/1`.

[16] R. Freitas and L. Chiu, "Solid-State Storage: Technology, Design and Applications." FAST2010 Tutorial, `http://www.usenix.org/events/fast10/tutorials/T2.pdf`, 2010.

[17] Transaction Processing Performance Council (TPC), TPC BENCHMARK C, Standard Specification,Revision 5.10, April 2008.

[18] R. Konishi, Y. Amagai, K. Sato, H. Hifumi, S. Kihara, and S. Moriai, "The Linux implementation of a log-structured file system," Operating Systems Review, vol.40, no.3, pp.102–107, 2006.

[19] SystemTap. `http://sourceware.org/systemtap/`.

[20] S. Boboila and P. Desnoyers, "Write endurance in flash drives: Measurements and analysis," FAST, pp.115–128, 2010.

[21] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," ACM Comput. Surv., vol.37, no.2, pp.138–163, 2005.

[22] Intel, "Intel SSD Optimizer, White Paper." `http://download.intel.com/design/flash/nand/mainstream/Intel_SSD_Optimizer White_Paper.pdf`.

[23] M. Rosenblum and J.K. Ousterhout, "The Design and Implementation of a Log-Structured File System," ACM Trans. Comput. Syst., vol.10, no.1, pp.26–52, 1992.

[24] J.M. Neefe, D.S. Roselli, A.M. Costello, R.Y. Wang, and T.E. Anderson, "Improving the Performance of Log-Structured File Systems with Adaptive Methods," SOSP, pp.238–251, 1997.

[25] A. Kawaguchi, S. Nishioka, and H. Motoda, "A Flash-Memory Based File System," USENIX Winter, pp.155–164, 1995.

[26] A. Ban, "Flash file system." US Patent No. 5404485, April 1995.

[27] J. Kim, J.M. Kim, S.H. Noh, S.L. Min, and Y. Cho, "A space-efficient flash translation layer for CompactFlash systems," IEEE Transactions on Consumer Electronics, vol.48, no.2, pp.366–375, May 2002.

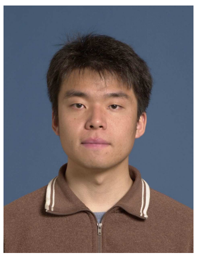
[28] JFFS2, The Journalling Flash File System, Red Hat Corporation, `http://sources.redhat.com/jffs2/`, 2001.

[29] YAFFS, Yet Another Flash File System, `http://www.yaffs.net`.

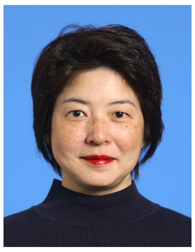
[30] G.J. Kim, S.C. Baek, H.S. Lee, H.D. Lee, and M.J. Joe, "LGeDBMS: A Small DBMS for Embedded System with Flash Memory," VLDB, pp.1255–1258, 2006.

[31] N. Agrawal, V. Prabhakaran, T. Wobber, J.D. Davis, M.S. Manasse, and R. Panigrahy, "Design Tradeoffs for SSD Performance," USENIX ATC, 2008.
[32] D. Myers, On the Use of NAND Flash Memory in High-Performance Relational Databases. Master's thesis, MIT, 2007.
[33] Y. Wang, K. Goda, M. Nakano, and M. Kitsuregawa, "Early Experience and Evaluation of File Systems on SSD with Database Applications," IEEE NAS, pp.467–476, July 2010.

**Yongkun Wang** is currently a Ph.D. candidate at the Graduate School of Information Science and Technology, the University of Tokyo in Japan. His research interests are the optimization of database systems in a storage platform with new storage media, specially the flash memory. He is a student member of DBSJ.



**Kazuo Goda** Project research associate at Institute of Industrial Science, the University of Tokyo in Japan. He received the Ph.D. degree in information science and technology from the University of Tokyo in 2005. His research interests include database systems and storage systems. He is a member of ACM, IEEE CS, USENIX, IPSJ and DBSJ.



**Miyuki Nakano** Project Associate Professor at Institute of Industrial Science, the University of Tokyo in Japan. She received the Ph.D degree in Information Science and Technology in 2006 from the University of Tokyo. Her research interests include data engineering and parallel computing. She is a member of IEEE, ACM, IEICE, IPSJ and DBSJ.

**Masaru Kitsuregawa** Professor and the director of the Center for Information Fusion at Institute of Industrial Science, the University of Tokyo in Japan. He received the Ph.D degree in information engineering in 1983 from the University of Tokyo. His research interests include parallel processing and database engineering. He is a member of steering committee of IEEE ICDE and Asian Coordinator for IEEE TKDE, and has been a trustee of the VLDB Endowment. He is currently a trustee of DBSJ. Recently he was awarded ACM SIGMOD E. F. Codd Innovation Award.