DEIM Forum 2009 E7-5

A Performance Study of Non-In-Place Update Based Transaction Processing on NAND Flash SSD

Yongkun WANG†, Kazuo GODA†, and Masaru KITSUREGAWA†

† Institute of Industrial Science, the University of Tokyo
4–6–1 Komaba, Meguro–ku, Tokyo 153–8505 Japan
E-mail: †{yongkun,kgoda,kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract NAND flash memory is one of the most aggressively scaled technologies among electronic devices recently. The massive increase in the capacity makes flash memory possible for enterprise applications, such as the database system. However, the characteristic of erase-before-write makes flash memory very challenging for the database management. A non-in-place update technique may be exploited to overcome the erase-before-write problem. In this paper, we provide a careful performance study of utilizing the non-in-place update technique for transactional database applications running on the flash memory. We deliberately design the experiment of the enterprise database system on flash memory, with a careful study on the performance gain, and reveals some insights into the enterprise applications built on the flash memory. In particular, we carefully tune the system to gain the superiority to the existing ones. We describe experiments in detail showing the benefits of utilizing non-in-place update on flash memory.

Key words NAND Flash Memory, SSD, LFS, Transaction Processing

1. Introduction

Flash memory is a kind of EEPROM (Electrically Erasable Programmable Read-Only Memory), invented at Toshiba in 1980 [1]. With many favorable features such as non-volatile, lightweight, high speed on read and write, it is widely used as the storage media for digital camera and portable devices, such as MP3/MP4 player, PDA, 3G Mobile Phone,etc.

According to the difference of the connections to the individual memory cells, the flash memory can be divided into two types: NOR and NAND. NOR allows random-access on a bit-wise basis, while NAND must be read on a block-wise basis, with typical block sizes of hundreds to thousands of bits. NAND flash memories are accessed much like block devices such as the hard disk, and thus viewed as a potential replacement of the hard disk. In this paper, we refer NAND flash memory as flash memory hereafter.

Recently, the flash memory is one of the most aggressively scaled technologies among electronic devices. Along with the decreasing of the price, the large capacity flash memory is now starting to appear on the market. flash memory SSD (Solid State Drive) begins to be equipped in the laptop. Compared to the traditional magnetic hard disk, the flash memory SSD is lightweight, noise free, vibrate resistant, temperature endurable and no mechanical parts, which make it a very promising storage media for computer system.

The massive increase in the capacity also makes flash memory available for enterprise applications, such as the database system. However, so far as some time-critical enterprise application is concerned, the research is still limited. When it comes to the enterprise application, the main advantage of the flash memory is the read performance. Obviously, however, the disadvantage is the out-of-place update, resulting in the ineffective performance of random write, especially the small size frequently random write. This disadvantage is fatal to those critical applications such as OLTP running on the bank system. For this reason, any rash behaviors on the critical enterprise application to hug the new technology without carefully considering on the drawbacks will eventually result in the disaster.

It is clear that the flash memory is entering the enterprise storage field gradually. A thorough understanding about the performance of enterprise applications on flash memory is a very pressing matter of the moment. A lot of literatures have reported the performance about flash memory in current computer system. However, in the case of enterprise database systems, especially the OLTP, it may not be a good choice to make direct use of flash memory, as shown by our experiment in Figure 5, and reported in [15]. The performance of database system is not improved significantly af-

ter simply replacing the hard disk with flash memory, even worse under the workload with highly intensive transactions. Therefore, people expect better techniques to fully utilizing the flash memory.

In this paper, we carefully study the out-of-place update characteristics of the flash memory, and provide the performance analysis for the enterprise database system running on the flash memory SSD. Our contribution can be summarized as follow:

- We summarize the benefits and potential problems of flash memory for enterprise applications such as the database system.
- We analyze the non-in-place update technique, introducing three rational methods of applying the non-in-place update technique to the flash memory in a systematic view. Besides, we point out the possible problem to be considered necessarily when utilizing this technique.
- We make a performance study of enterprise database system on flash memory in terms of the non-in-place update, using the TPC-C benchmark to generate the comprehensive measurements of transaction throughput. The concrete experiments are designed with an enterprise database product. We reveal the superiority of non-in-place update on flash memory by carefully tuning the experiment system.

The rest of this paper will organize as follow: Section 2 will present the benefit and issue of flash memory for transaction processing system. In Section 3, we will introduce the non-in-place update based database system on flash memory. The deliberately designed experiment will be described in Section 4. Section 5 will summarize the related work. Finally, our conclusion and the future work will be provided in Section 6.

2. Issue of Flash Memory with Transaction Processing

Unlike the traditional hard disk, which has an approximately symmetric read and write speed, flash memory, on the contrary, has a substantial variation in the speed of read and write, as shown by our raw device test in Table 1. The average response time of read, whatever in sequential or random mode, as well as that of the sequential write, is about two orders of magnitude faster than that of the hard disk. Instead, the average response time of the write in random mode, is comparable or even worse than that of the hard disk. This is primarily because the flash memory cannot be updated in place, a time-consuming block erase operation having to be performed before the write operations, as disclosed in Table 2 [11]. For the sake of better performance, the size of erase block is usually large, about several hundred KB, leading to a expensive time cost of erase operation compared to that of flash read.

Table 1 Average response time (μs) of the flash memory SSD and hard disk with the transfer request size of 4KB. Experiment setup is the same as that in Section 4. 1 except here the hard disk and flash memory SSD is used as a raw device. Benchmark program is Iometer 2006.07.27 [19].

	Hard Disk		Flash Memory SSD	
	Read	Write	Read	Write
Sequential	127	183	94	75
Random	13146	6738	106	8143

Table 2 Operational flash parameters of Samsung 4GB flash memory chip

Page Read to Register	$25\mu s$
Page Program (Write) from Register	$200\mu s$
Block Erase	$1.5ms$
Serial Access to Register (Data bus)	$100\mu s$

The ineffective random write performance of flash memory could be painful for some transaction processing applications, such as the enterprise database systems, specially the OLTP. In these applications, the intensive random write is sometime the main stream of disk I/O. Though the operating system has an efficient buffer policy to cache the individual write operations into a bulk update, some critical applications such as the bank system enforce the data to be flushed into disk to ensure the safety of data. Therefore, it would be problematic for the existing transaction processing systems to run on the flash memory directly, as reported in [15]. Our experiment also illustrates this points in Figure 5 that the performance is not improved, even worse than that of the hard disk sometime by directly using the flash memory SSD as the main storage media of data, though the flash memory has a fine performance on read and sequential write. A better solution is required to fully exploit the benefit of flash memory SSD.

By far, the research of improving the random write performance on flash memory focuses on reducing the time caused by the erase operations when data is being updated frequently. A non-in-place update technique is introduced to put off the erase operations as long as there is free data blocks on flash memory. Hence, exploiting the non-in-place update on enterprise system could be a promising alternative on flash memory. In the next section we will not only analyze the reason of the improvement but summarize the possible methods of utilizing the non-in-place update technique on flash memory.

3. Non-In-Place Update Based Database System on Flash Memory

As described in previous section, the performance is not

improved to utilize the flash memory directly for enterprise database system. A tactful way is to introduce the non-in-place update for enterprise system on flash memory to improve the overall performance. The idea of the non-in-place update technique, arisen from the log-structured file system described in [2], with an implementation called *Sprite LFS*. Instead of seeking and updating in-place for each file and Inode, the LFS will collect all write operations and write them into a new address space continuously. The principal feature of LFS is that a large number of data blocks are gathered in a cache before writing to disk in order to maximize the throughput of collocated write operations, thereby minimizing seek time and accelerating the performance of writes to small files. Though log-structured file systems optimize write performance by some detriment of scan performance [10], this feature is greatly helpful on flash memory to make up for the inefficient random write performance since the read performance is about two orders of magnitude higher than that of erase operations. The overall write performance is hereby improved.

Therefore, the non-in-place update based database system on flash memory could have potential performance gain. In this case the flash memory is usually written sequentially through all the way, with a background process reclaiming the obsolete data blocks into the pool of available data block. On the basis of non-in-place update, all the update operations are performed by writing the data pages into the new flash pages, and the erase operations are not required right beforehand as long as the free flash pages are available. Thus, the overall throughput of transactions can be improved along with the step-up of I/O by reducing the instantly erase operations. Though some detriment of read performance, providing that the read performance of flash memory is about two orders of magnitude faster than that of the erase operations.

The key design of non-in-place update based system can be seen in Figure 1. Here the free space management is implementated by dividing the storage media into storage units called *segment*, which is usually much bigger than the size of block using by the operating system. Within each segments, the data and Inode are written sequentially like a log. A special data structure called *Superblock* will be stored in the fixed area, to hold the static configuration information such as number of segments and segment size. New data blocks of files and directories along with Inode are written sequentially in each segment, as shown in Figure 2. When updating a data block, the data block is read into cache from its segment, and applied the changes, then written to a new segment with other data blocks together. The old data block will be obsolete and can be cleaned in the near future by a daemon process.

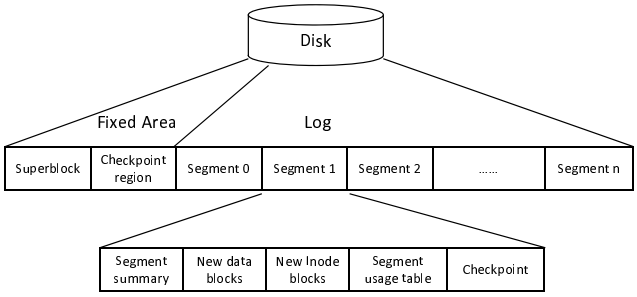

Fig. 1 Managing the free space by segments in Log-structured file system

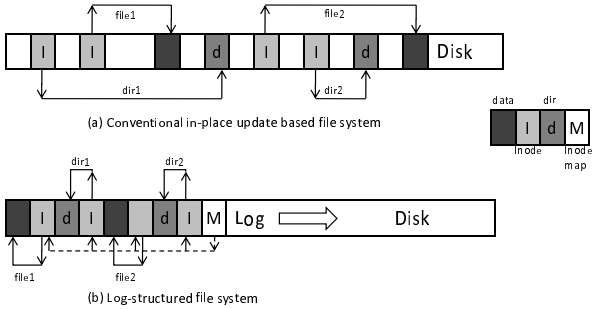

Fig. 2 Files and Directories are written sequentially in Log-structured file system

From a macro view of system, there are several possible methods to utilize the non-in-place update on flash memory, described as follow:

- *Traditional DBMS on traditional file system on Log-structured FTL*
- *Traditional DBMS on log-structured file system on simple-mapping FTL*
- *DBMS with log-structured storage access implementation on traditional file system on simple-mapping FTL*

All of these methods could be successful solutions to reach high throughput of database transaction processing on flash memory. In this paper, we only have the performance study and experiment evaluation on the second method. Evaluation on the rest methods is beyond the scope of this paper, and possibly provided in the later literatures by us with further research.

It is to be noted here that a concern on the non-in-place update technique is the design and settings of GC (Garbage Collection). Since the non-in-place update technique consumes free flash pages faster than other methods, the obsolete data pages (garbage) should be reclaimed by fine timing policy to the pool of available data blocks to ensure there is free pages available anytime when there are write requests. The GC strategy will have direct influence to the overall performance of system. A better GC strategy can emulate to the upper level system that the free data blocks are always available, with minimum cost of CPU time.

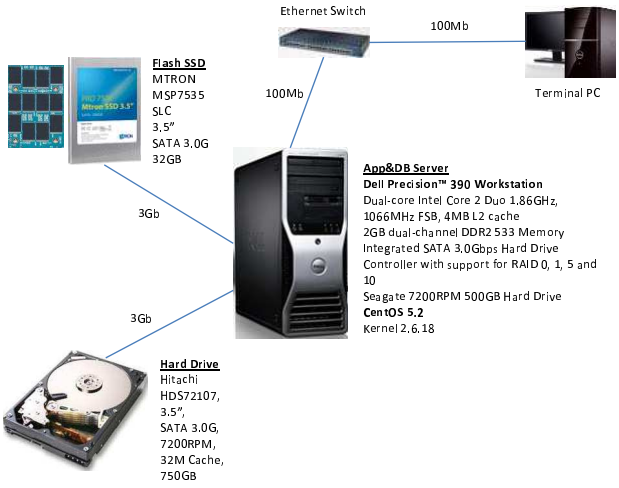

Fig. 3 System Configuration

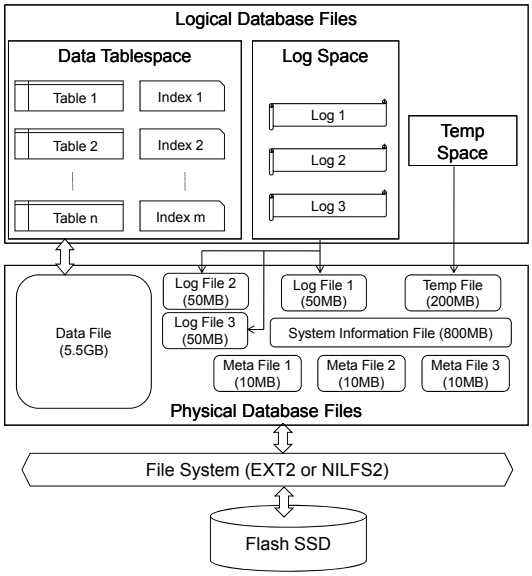

Fig. 4 System Storage Hierarchy on Flash Memory SSD

We will discuss the influence of the GC settings in our experiment described in next section.

4. Experimental Evaluation

We now describe a set of experiments that validate the efficiency of the non-in-place update technique and compare them against the traditional alternative. We use the popular TPC-C [16] as the benchmark, though it cannot exactly emulate the real production workload [17], it discloses the general business process and workload, supported by the main hardware and software database system providers in the industry. The main experiment, transaction throughput of TPC-C benchmark, illustrates the considerable improvement on flash memory. In addition, we discuss the influence of GC in our experiment with a test case.

4.1 Experiment Setup

We build a database server on an enterprise level Linux system, with dual-core Intel Core 2 Duo 1.86GHz (1066MHz FSB, 4MB L2 cache, 64-bit) CPU and 2GB RAM. The flash memory SSD is MTRON MSP7535 32GB SLC flash memory, connected to the computer system with SATA 3.0Gbps hard drive controller. Figure 3 gives the view of our experimental system.

As for the basic settings of the database system in our case, the buffer cache is 8MB, with a 4KB block size. The block size is set by our previous empirical experiment, in which we performed a low level disk IO test, with a raw device test program written by us. Therefore, we choose the optimal request size as the DB block size which is around 4KB.

In order to verify the non-in-place update technique on flash memory, we choose a traditional log-structured file system, NILFS2 [8] [18], as an intermediate layer between the DBMS and flash memory SSD. As a comparison, we choose the EXT2 file system as the representative of a fully-fledged file system running on the Linux server system.

The storage hierarchy is shown in Figure 4. Above all, we format the flash memory SSD with EXT2 file system, on which we build the database instance, with all the related files together, such as the data files and log files. Thus, all the activities of this instance are confined within the flash memory SSD. We refer this system as "EXT2", as well as taking it as the suffix in the figures of the subsequent part of this paper. Similarly, we format the flash memory SSD with NILFS2, on which we build the same instance as EXT2 system. We refer this system as "NILFS2" hereafter.

4.2 Transaction Throughput

Transaction throughput is an important measure of performance for database system. In this test, we use threads to simulate the virtual users. Each virtual users will have a dedicated warehouse during the execution of transactions. Unlike the real user, virtual users in our test do not have the time for "Key and Think", for the purpose of getting intensive transaction workload. We gradually increase the number of warehouses as well as the number of virtual users to match. The result of transaction throughputs is shown in Figure 5. Here the *tpm* denotes the transactions-per-minute, in which the performance metric is expressed by counting in all types of transactions. It can be seen that the transaction throughput on flash with EXT2 file system is comparable, even worse than that on hard disk with EXT2, which verifies our perspective that it is not beneficial for transactions-intensive applications by directly utilizing the flash memory. Remarkably, a significant improvement of the transaction throughput can be found on flash with NILFS2: the transaction throughput is up to 7 times higher than that on flash with EXT2, which manifests that non-in-place update based transaction processing system can undergo dramatic

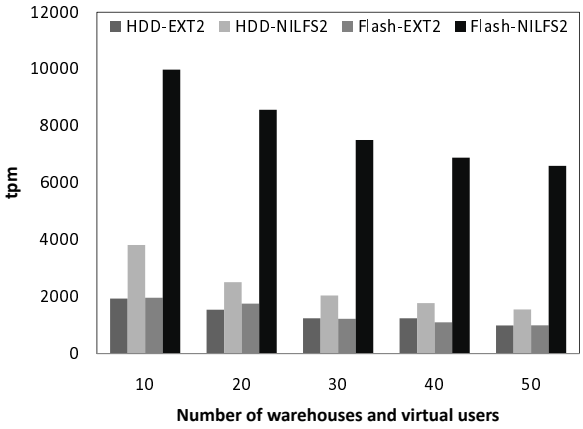

Fig. 5 Comparison of transaction throughput in tpm on hard disk (HDD) and flash memory SSD (Flash) with different file system. The transaction throughput on flash memory SSD with NILFS2 (denoted by Flash-NILFS2, which is non-in-place update based) is up to 7 times higher than that with EXT2 which is in-place update based.

improvements on flash memory.

4.3 IOPS

IOPS (Input/Output operations Per Second) is a important measure used by I/O measurement applications such as Iometer [19]. Generally, the IOPS can help us to better understand the performance of I/O system. The specific number of IOPS varies greatly depending upon the variables the tester enters into the program, such as the queue length and data block sizes. In our test, we examine the total number of transfers per second that were issued to physical devices. A transfer is an I/O request to a physical device. Multiple logical requests can be combined into a single I/O request to the device. So a transfer is of indeterminate size. Our trace result is shown in Figure 6, in which the IOPS denoting the I/O request per second. As disclosed in Figure 6, the total IOPS of EXT2, either on hard disk or flash memory, is around 400. By comparison, the total IOPS on flash memory with NILFS2 is outstanding: up to 6 times higher than that on EXT2. It implies that the non-in-place update system can handle more requests at a time.

Besides the number of I/O requests per second, Figure 7 shows the average time for I/O requests issued to the device to be served, which includes the time spent by the requests in queue and the time spent servicing them. When the number of warehouses and virtual users is small, the I/O request is combined by short and frequent reads and writes, the service time of NILFS2 is only about one-fourth of that of EXT2. Though along with the increase of warehouses and virtual users, the I/O request tends to become longer, the non-in-place update based system is still superior over the in-place update based system. Since the response time is

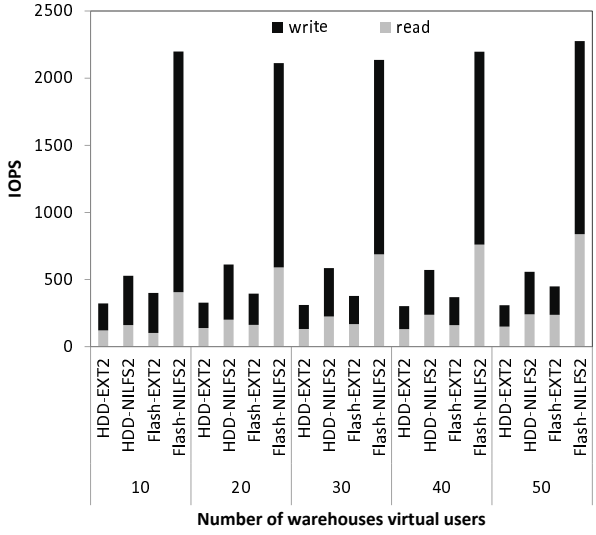

Fig. 6 IOPS

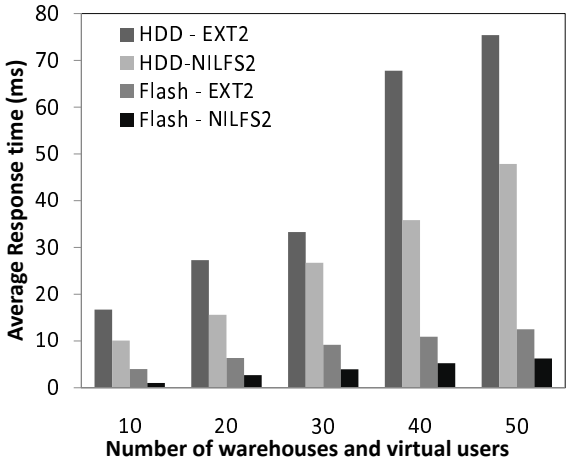

Fig. 7 Average Response Time

cut down greatly by non-in-place update, the OLTP applications, which is required to respond immediately to user requests, could be benifited a lot.

4.4 CPU Utilization

In this section we discuss the CPU Utilization in order to analysis the bottleneck of our experimental system. The CPU Utilization is monitored when the transactions running in the steady state. The startup and terminated effect is eliminated. We sampled 100 times during a period of around two-third of the whole execution time. Then the average of CPU Utilization is obtained and shown in Figure 8, in which the CPU Utilization is divided into four portions: *user, system, iowait* and *idle*, indicating the CPU time spent on user application, kernel, wait for I/O complete and idle. The main portion of CPU Utilization of hard disk and flash memory SSD on EXT2 is spent on wafting for the completion of I/O (iowait), which implies that the system is possibly "IO-Bound". However, the CPU Utilization of flash on NILFS2 contrasts strongly in the ratio of four portions with

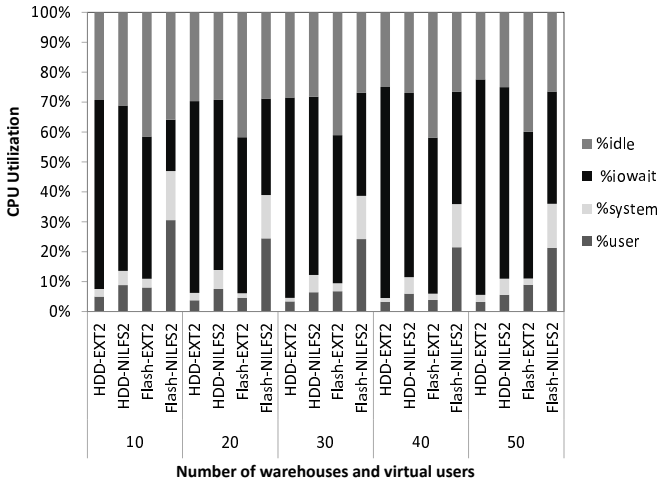

Fig. 8 CPU Utilization, non-in-place update based system (denoted by Flash-NILFS2) reduces the CPU time spent on waiting for I/O complete.

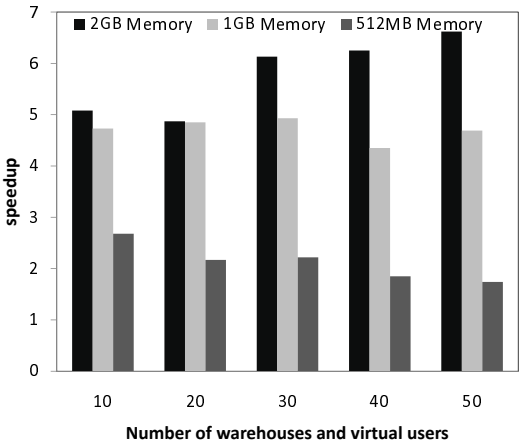

Fig. 9 Performance speedup with different amount of physical memory

the other cases: a uniform distribution of CPU time is observed, caused by the cutback of the CPU time spent on I/O wait, and balanced by more CPU time moved to running the user applications.

4. 5 Disk Buffer Cache

Although we have limited the buffer cache of the database system to a very small size, there is still some influence from the disk buffer cache, as long as we use the file system to manage the data blocks written to the disk drive. At this moment, we cannot eliminate the influence of system buffer cache like using a raw device. A negative but efficient approach is to test the system with bound physical memory. Figure 9 shows the result with 1GB and 512MB physical memory in the same experiment system described in Section 4. 1. Though the available system memory is reduced sharply, the speedup of performance by the non-in-place update technique is still above 4 and around 2 respectively, which demonstrates the superiority of the non-in-place update technique under the extreme conditions.

4. 6 Influence of Garbage Collection

We now discuss the influence introduced by the different settings of GC on NILFS2. In Section 4. 2, no cleaning occurs during the execution, so the measurements represent best-care performance.

In order to protect the data for recovery, in NILFS2 file system, the recent checkpoints are protected from GC during the period given by a GC parameter "protection period"; GC never deletes checkpoints whose age from its creation time is less than the value given by the protection period. The default protection period is 3600 seconds with 5 seconds cleaning interval, in this condition the snapshot will eat up the disk space by the writes of highly intensive transactions. The disk becomes full, although lots of data blocks are not valid and should be reclaimed.

Once the GC is enabled for reclamation, our main concern is whether it has a remarkable influence on the performance of transaction processing. According to the design of NILFS2, the following settings might have influence on the performance, which also apply to other log-structured file system generally.

- The *protection period*. The data blocks whose age is longer than this period will be cleaned. If this period is too long, the old data blocks will not be cleaned in time. As a result, the free data blocks will be eaten up quickly, which will cause frequently erase operations for the new coming data. Therefore, the advantage of the non-in-place update technique will lost. On the contrary, if this period is too short, the CPU will be busy with the cleaning of the old data blocks and snapshots. Consequently, the bandwidth of the system will be wasted. In a word, a better policy on this setting is required.
- The *number of segments per clean*. This setting denotes how fast the cleaning is. With larger number, the cleaning can be performed quickly, with higher CPU cost.
- The *cleaning interval*. This is the time interval between the GC is invoked to collect the obsolete data according to the protection period. Too long or too short cleaning interval will have the same influence as the protection period.

Actually, so far there is no appreciable change in the result compared with that without GC after we choose a more greedy GC setting and redo the experiments. Table 3 shows the experimental results of the test with a greedy GC settings of $(1, 2, 5)$, in the sequence of (protection period, number of segments per clean, cleaning intervals). We will go further on exploring the influence under various GC settings.

Table 3 Performance Metrices of NILFS2 based transaction throughput on flash memory with GC for 10 warehouses and 10 virtual users.

	tpm	**IOPS**	**Average Response Time (ms) of I/O Request**
No GC	9983	reads: 406 writes: 1792 total: 2198	1.02
GC(1, 2, 5)	9933	reads: 384 writes: 1856 total: 2240	1.82

5. Related Work

Our study draws on the concepts and principles in traditional log structured file systems and flash-based technologies. In this section, we discuss the relevant examples of prior work in these areas.

5.1 LFS

Sprite LFS [2] introduced the concept of log-structured file system design. Instead of seeking and updating in-place for each file and Inode, the LFS collects all write operations and write them into a new address space continuously. Though log-structured file systems optimize write performance to the detriment of scan performance [10], this feature is greatly helpful on flash memory to make up for the inefficient random write performance since the read performance is about two orders of magnitude higher than that of erase operations.

5.2 Flash-based Technologies

By a systematical "Bottom-Up" view, the research on flash memory can be categorized as follow:

Hardware Interface This is a layer to bridge the operating system and flash memory, usually called FTL (Flash Translation Layer). 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 using a simple but efficient page-to-page mapping [5] with a log-structured architecture [2]. However, it requires a lot of space to store the mapping table. In order to reduce the space for mapping table, the block mapping scheme is proposed, using the block mapping table with page offset to map the logical pages and flash pages [6]. However the block-copy may happen frequently. To solve this problem, Kim improve the block mapping scheme to hybrid scheme by using a log block mapping table [7].

File System Most of the file system designs for flash memory are based on Log-structured file system [2], as a way to compensate for the write latency associated with erasures. JFFS, and its successor JFFS2 [3], are journaling file systems for flash. The JFFS file systems are not memory-efficient for storing volatile data structures, and require a full scan to reconstruct these data structures from persistent storage upon a crash. JFFS2 performs wear-leveling, in a somewhat ad-hoc fashion, with the cleaner selecting a block with valid data at every 100th cleaning, and one with most invalid data at other times. YAFFS [4] is flash file system for embedded devices. It treats handling of wear-leveling akin to handling bad blocks, which appear as the device gets used.

Database System Previous design for database system on flash memory focused on the resource-constraints environment such as the embedded systems or sensor networks, in a log-structured behavior. FlashDB [12] is a self-tuning database systems optimized for sensor networks. It can switch between two modes: disk mode, much like regular B+ trees, using for frequently read or infrequently write; log mode, employed a log-structured approach, using for frequently write. LGeDBMS [13], is a relational database system for mobile phone.

For enterprise database design on flash memory SSD, In-Page Logging [14] is proposed. The key idea is to co-locate a data page and its log records in the same physical location.

Our work belongs to the system design on flash memory, on the basis of which, we are engaged in researching on the solutions for enterprise database system with complicated transaction management, by exploiting a non-in-place update technique for storage.

6. Conclusion and Future Work

For transaction processing system on flash memory, we describe a non-in-place update technique to improve the transaction throughput. In a non-in-place update based system, the write operations are performed sequentially; while the GC cleans the obsolete data in the background. This strategy greatly reduces the frequency of time-consuming erase operations for applications with intensive write operations, thereby resulting in improved overall performance, espeically the transaction throughput. We use a traditional log-structured file system to build a test model for examination. We then validated this technique with a set of experiments and showed that the non-in-place update based system can considerably speed up the transaction throughput on flash memory.

In the near future, we plan to apply the non-in-place update technique into different layers of the system and investigate appropriate algorithms for different context. In particular, since there are many special factors of the transaction processing system having influence over the performance, we would like to take these factors into account for the design of experiment next time.

References

[1] Fujio Masuoka, US patent 4531203, http://v3.espacenet.com/textdoc?DB=EPODOC&IDX=US4531203

[2] M. Rosenblum and J. Ousterhout. The Design and Implementation of a Log-Structured File System. ACM Transactions on Computer Systems, 10(1):26C52, 1992.

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

[4] C. Manning. YAFFS: Yet Another Flash File System. http://www.aleph1.co.uk/yaffs, 2004.

[5] Atsuo Kawaguchi, Shingo Nishioka, and HiroshiMotoda. A Flash-memory based file system. In USENIX Winter, pages 155C164, 1995.

[6] BAN, A. 1995. Flash file system. United States Patent, No. 5,404,485 (Apr.).

[7] KIM, J. S., KIM, J.M.,NOH, S.H.,MIN,S.L., AND CHO, Y. K. 2002. A space-efficient Flash translation layer for compact Flash systems. IEEE Trans. Cons. Elect. 48, 366C375.

[8] NTT, New Implementation of a Log-structured File System, http://www.nilfs.org/en/about_nilfs.html

[9] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, Rina Panigrahy, Design Tradeoffs for SSD Performance, 2008 USENIX Annual Technical Conference

[10] Goetz Graefe. Write-Optimized B-Trees. In Proceedings of the 30th VLDB Conference, pages 672-683, Toronto, Canada, September 2004.

[11] Samsung Corporation. K9XXG08XXM Flash Memory Specification. http://www.samsung.com/global/system/business/semiconductor/product/2007/6/11/NANDFlash/SLC_LargeBlock/8Gbit/K9F8G08U0M/ds_k9f8g08x0m_rev10.pdf, 2007.

[12] Suman Nath, Aman Kansal: FlashDB: dynamic self-tuning database for NAND Flash. IPSN 2007:410-419

[13] Gye-Jeong Kim, Seung-Cheon Baek, Hyun-Sook Lee, Han-Deok Lee, Moon Jeung Joe: LGeDBMS: A Small DBMS for Embedded System with Flash Memory. VLDB 2006:1255-1258

[14] Sang-Won Lee, Bongki Moon: Design of Flash-based DBMS: an in-page logging approach. SIGMOD 2007:55-66

[15] Andrew Birrell, Michael Isard, Chuck Thacker, Ted Wobber: A design for high-performance Flash disks. Operating Systems Review (SIGOPS) 41(2):88-93 (2007)

[16] Transaction Processing Performance Council (TPC), TPC BENCHMARK C, Standard Specification, Revision 5.10, www.tpc.org

[17] Windsor W. Hsu, Alan Jay Smith, Honesty C. Young: Characteristics of production database workloads and the TPC benchmarks. IBM Systems Journal (IBMSJ) 40(3):781-802 (2001)

[18] Ryusuke Konishi, Yoshiji Amagai, Koji Sato, Hisashi Hifumi, Seiji Kihara, Satoshi Moriai: The Linux implementation of a log-structured file system. Operating Systems Review (SIGOPS) 40(3):102-107 (2006)

[19] Iometer, http://www.iometer.org/