DEIM Forum 2010 E8-2

An Experimental Study on IO Optimization Techniques for Flash-based Transaction Processing Systems

Yongkun WANG[†], Kazuo GODA[†], Miyuki NAKANO[†], 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,miyuki,kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract Flash SSDs are being considered and partially starting to be utilized for enterprise storage. In order to maximize the performance benefit, different IO optimization techniques can be applied to the existing storage system. We examined the IO optimization techniques and the distinct features of the flash SSD. The IOs applied with optimization techniques are analyzed through the IO path with the trace generated from the transaction processing system which is usually hosted on the enterprise storage platforms.

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

1. Introduction

Flash SSDs (Solid State Drive) are being considered and partially starting to be utilized for enterprise storage. Many enterprise storage platforms, such as EMC Symmetrix V-Max [2], Oracle EXADATA V2 [11], have incorporated flash SSDs to boost the IO performance of the whole system. However, the flash SSD is a brand new storage media, with special IO characteristics which is far away from the traditional hard disk, such as the fast read performance, "erase-before-write", and wear-leveling. Therefore, the conventional IO optimization techniques should be reconsidered based on the characteristics of flash SSD.

IO optimization techniques are important to the overall performance of the storage system, especially the transaction processing system. The IO optimization techniques are usually designed by the characteristics of the workload and storage media. In the case of the transaction processing systems, the workload is mainly composed of the random writes to the storage media. For the widely used storage media, hard disk, the random writes are very slow due to the mechanical moving parts. Hence lots of IO optimization techniques are applied to try to convert the random writes to sequential writes in order to maximize the overall performance. For example, at file system level, the log-structured file system is proposed to convert the random writes to sequential writes, with the side-effect that sequential reads may also be converted into random reads. At the block IO level, the IO scheduler plays a vital role to group, merge and re-order the requests to utilize the sequential performance of the storage media.

For flash SSD, these IO optimization techniques should be reconsidered. For example, for log-structured file system, although the sequential reads may be converted into random reads, the performance may not be harmed since the performance of sequential read is close to that of random read on flash SSD since there is no mechanical moving parts.

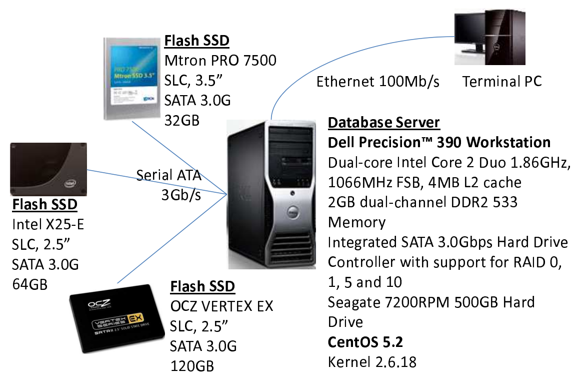
Therefore, we studied the IO optimization techniques and the distinct features of the flash SSD. IOs applied with optimization techniques are analyzed through the IO path with the trace generated from the transaction processing system which is usually hosted on the enterprise storage platforms.

The rest of this paper will organize as follow: Section 2 will describe the experiment setup. In Section 3, we will present the transaction throughput. The trace-based analysis will be provided in Section 4. Section 5 will summarize the related work. Finally, our conclusion and the future work will be provided in Section 6.

2. Experiment Setup

Our experiments focus on the transaction processing system, which is one of the most important applications hosted on enterprise storage platform. We used the popular TPC-C [15] as the benchmark for the performance of transaction processing systems. A database server to run the TPC-C benchmark was built with the Linux operating system. High-end SLC flash SSDs are connected to the computer system with SATA 3.0Gbps hard drive controller. These SSDs are different in random access time or throughput. Fig. 1 gives the view of our experimental system.

We chose a commercial DBMS, as well as popular open source DBMS MySQL, as the database system for the TPC-



Flash SSD
Mtron PRO 7500
SLC, 3.5"
SATA 3.0G
32GB
Ethernet 100Mb/s
Terminal PC
Database Server
Dell Precision™ 390 Workstation
Dual-core Intel Core 2 Duo 1.86GHz, 1066MHz FSB, 4MB L2 cache
2GB dual-channel DDR2 533 Memory
Integrated SATA 3.0Gbps Hard Drive Controller with support for RAID 0, 1, 5 and 10
Seagate 7200RPM 500GB Hard Drive
CentOS 5.2
Kernel 2.6.18
Serial ATA 3Gb/s
Flash SSD
Intel X25-E
SLC, 2.5"
SATA 3.0G
64GB
Flash SSD
OCZ VERTEX EX
SLC, 2.5"
SATA 3.0G
120GB

Fig. 1 System Configuration

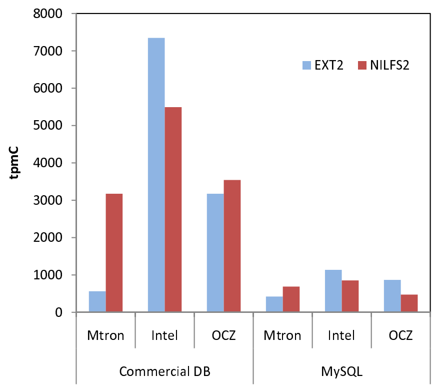


Fig. 2 Transaction Throughput with "Anticipatory" scheduler with 30 users and 30 warehouses

C benchmark.

In the commercial database system, the buffer cache was set to 8MB, redo log buffer was 5MB, and the block size was 4KB. For logging, we set the behavior to immediately do flushing of the redo log buffer with wait when committing the transaction. The data file was fixed to a large size in order to have a better sequential IO performance. All the IOs were set to be synchronous IO, but not direct IO since we use file system.

For MySQL, we installed the *InnoDB* storage engine. The data buffer pool size was 4MB, log buffer was 2MB, and the block size was 16KB. The block size of MySQL was different from that of the commercial DBMS, because MySQL does not allow us to configure the block size, although 16KB might not be optimal. We also fixed the size of data file instead of "autoextend". Synchronous IO behavior was chosen. For the flushing method of log, we set it to flush the log at transaction commit; for the flushing of data, we used the synchronous IO.

Different file systems have different optimizations on the IO. We evaluated two file systems on flash SSD, the conventional EXT2 file system and a log-structured file system, NILFS2 [7].

For EXT2, we set it with the default block size and group size, that was 4KB blocks and 32K blocks per group.

For NILFS2, we set the block size to 4KB too, with 2KB blocks per segments. The segment size is influential to the garbage collection (GC, a.k.a segment cleaning) policy. By default we disabled it for the simplicity of analysis.

The "Anticipatory" was chosen as the default IO scheduling algorithm. The IO scheduling is important for the traditional file system. For the log-structured file system, we think it is not necessary to do IO scheduling since the log-structured file system has already organized all the writes to sequential writes, while the read, both sequential and random, is fast and minor part of IO.

For the TPC-C benchmark, we started a number of threads to simulate virtual users, the number varying from 10 to 30, with 10 to 30 warehouses accordingly. The "Key and Thinking" time was set to zero in order to get a hot workload. The mix of the transaction types followed the standards in [15], that is, "New-Order": 43.48%, "Payment": 43.48%, "Order-Status": 4.35%, "Delivery": 4.35%, "Stock-Level": 4.35%. Only the "New-Order" transaction was counted in the transaction throughput since it is the backbone of the workload [15].

Some SSDs do not allow us to disable the cache, so the write-back cache was enabled on each SSD by default.

3. Transaction Throughput

We present the transaction throughput by special IO optimization technique. We will examine the following cases:

- Case 1: Data files are hosted on SSD formated with traditional in-place update file system
- Case 2: Data files are hosted on SSD formated with log-structured file system

We compare the performance in Case 1 and Case 2. The transaction throughput is shown in Fig. 2. We find that the log-structured file system is superior to traditional file system on Mtron SSD, but not outstanding on other SSDs. We think it is due to the different character of each SSD. Section 4.2 will discuss this point.

4. Performance Analysis

In this section, we will analyze the effect of the IO optimization techniques, combined with the prominent performance character of flash SSD.

4.1 IO Optimizations

We placed trace points at VFS layer, generic block IO (BIO) layer and IO scheduler layer, as shown in Fig. 3. Firstly, we examined the difference by the difference IO optimizations at file system level, that is, the different processing results between the traditional file system and log-structured

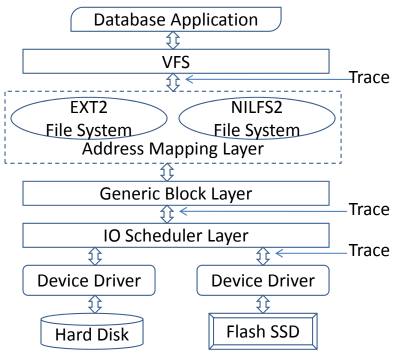


Fig. 3 IO path in the Linux kernel of our experiment system

file system. The amount of IOs at the VFS layer is almost identical for the two file system. A comparison between the VFS layer and generic block IO layer will tell us the changes after processed by different file systems. Fig. 4 shows the difference by the number of requests per transaction. Clearly, the number of read requests decreased a lot, due to the hit by file system buffer. A close view about the difference of writes is shown in Fig. 5, which shows that the number of write requests does not change on EXT2, while the number of requests reduces at 2x to 3.5x for NILFS2 in the case of MySQL, but it does not vary in the case of Commercial database system. We conjecture that it is due to the different buffer policy of different storage engine by the database applications. Fig. 6 shows the changes in bytes, which confirms that not only the number of requests, but the total amount of reads in bytes are absorbed by the buffer. A close view on writes shown in Fig. 7 tells us that the write amount in bytes are still not changed on EXT2, but it is increased on NILFS2, especially in the case of commercial database. Combined with the number changes of the write requests shown in Fig. 5, it is clear that each request is enlarged in bytes for commercial database on NILFS2, although the number of write requests is not changed. This shows that more writes are added on NILFS2 compared to the EXT2. More IOs than EXT2 will put the NILFS2 at a disadvantage.

We also analyzed the changes processed by the IO scheduler. As for the changes about the number of requests and the amount of bytes, Fig. 8 and Fig. 9 show that none of them has experienced a big change after processed by IO scheduler, which shows that the requests of transaction processing system are dominated by random requests which cannot be merged significantly.

Another function of the IO scheduling is the re-ordering. Fig. 10 shows the address distribution after processed by the IO scheduler. At this time, the sectors are ready for transfer via the device driver. Fig. 10(a) shows that the writes are organized to approximately sequential write on each run for commercial DB on EXT2, noticed that the sequential log

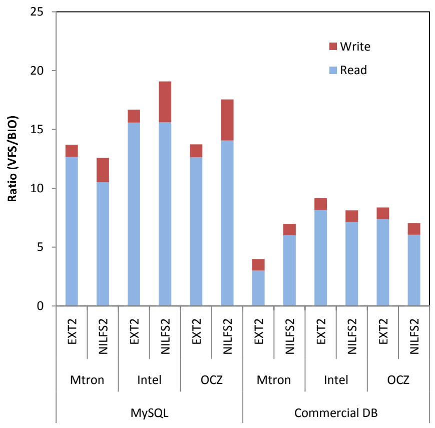


Fig. 4 Ratio between the VFS and generic block layer about the number of requests per transaction

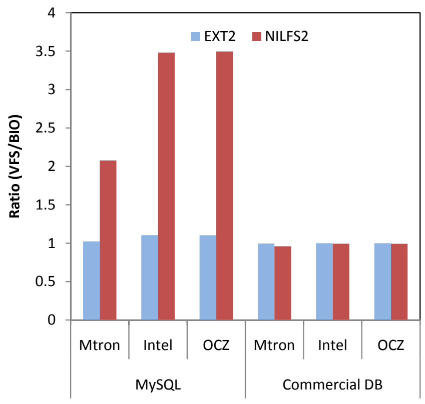


Fig. 5 Ratio between the generic block layer and VFS layer about the number of write request per transaction

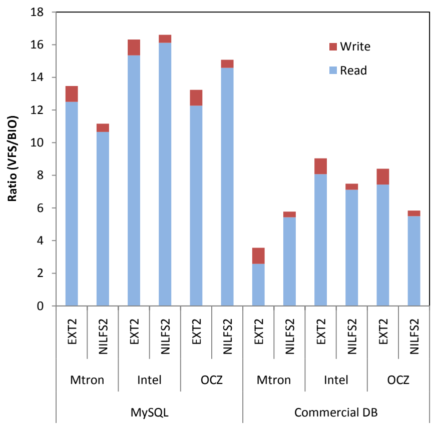


Fig. 6 Ratio between the VFS and generic block layer about the amount of IOs in bytes per transaction

writes are shown both at the top and bottom as a line. The reads are in burst between each run of writes. As comparison, Fig. 10(b) shows the access pattern of commercial DB on NILFS2: the writes are completely linear for all the data, and reads are in burst as well.

The address distribution in Figure 10 shows that the writes in the conventional file system are well ordered by the IO

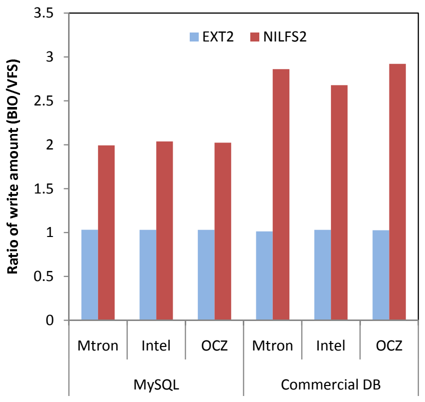


Fig. 7 Ratio between the generic block layer and VFS layer about the amount of writes in bytes per transaction

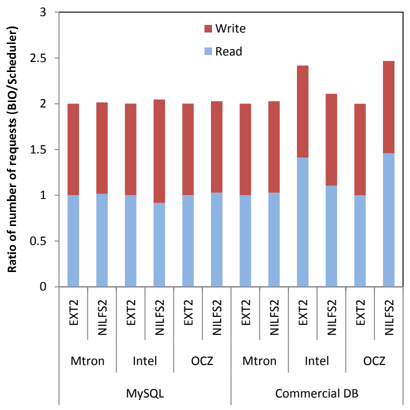


Fig. 8 Ratio between the generic block layer and IO scheduler layer about the number of requests per transaction

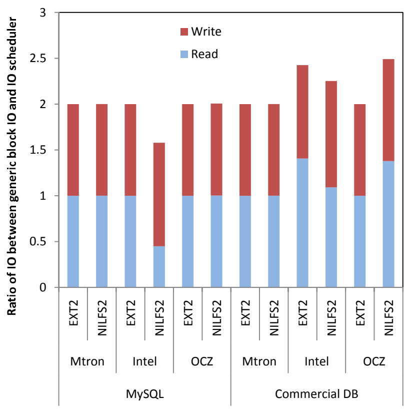
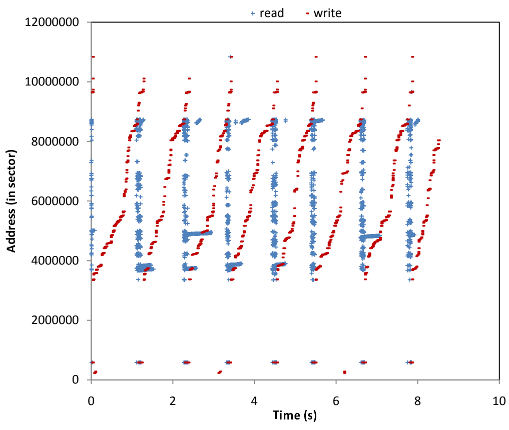
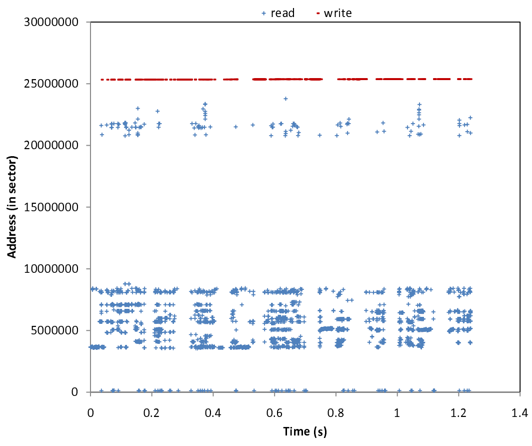


Fig. 9 Ratio between the generic block layer and IO scheduler layer about the amount of IOs in bytes per transaction

scheduler, so that it appears to be approximately linear in each run. However, the writes are limited in the address space of the file, rendering frequently going back and writing from the starting address of the file, causing the erase operations for each run. As a comparison, writes in log-structured file system are well organized and distributed to the whole disk space. We can see that the write addresses are always linear increasing providing there are enough disk



(a) Commercial DB on EXT2



(b) Commercial DB on NILFS2

Fig. 10 The sector address of Commercial database IOs transfered to Intel SSD by device driver after processed by "Anticipatory" scheduling algorithm. The sector address on other SSDs have similar access patterns and are not shown here for sake of brevity. Both figures contain equal number of IOs.

spaces. Consequently, the erase operations would happen much less than that on EXT2. Therefore, the address distribution shows that the flash SSD would be quite favorable on the log-structured file system providing the same number of IOs.

4.2 Prominent Performance Character of Flash SSD

We discuss the prominent performance character of flash SSD which will be favorable for the log-structured file system, and disclose the reason of the difference of transaction throughput on different file systems. Figure 11 show the performance gain of the sequential IO against random IO with one million requests for each size on "raw" SSD device. The read-ahead number is set to 256 sectors. Even with read-

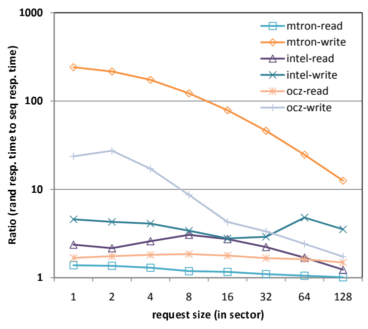


Fig. 11 Performance ratio of sequential IO to random IO

ahead, the gap between sequential and random reads is close to 1 for Mtron and OCZ SSD. We got the same results for these two SSDs even when we disabled the read-ahead. For Intel SSD, sequential read speedup ratio is round 2x to 3x . We observed in our test system that the amount of sequential reads of transaction processing system are small compared to the random reads, so the suffering from random reads on NILFS2 is small. We then mainly focus on the gap caused by the sequential writes to random writes. The "mtron-write" line shows that the sequential write speedup ratio of Mtron is more than 100 times for small request. But this speedup is obtained by the evenly distributed random writes on the whole disk spaces. Recall the address access pattern shown in Figure 10(a), the writes on EXT2 is approximately linear in each run within the file address space. That is why the overall transaction performance is only around 6x instead of $100/2 = 50$ times, where 2 comes from the two times writes on NILFS2 as many as that on EXT2. On Intel SSD, the gain of sequential to random is only around 2x to 3x. So the overall performance on NILFS2 is not better than that on EXT2 with twofold increase of the sectors to write. It is medium on OCZ SSD since the sequential speedup ratio is about 8x to 18x, which ranks between the Mtron SSD and Intel SSD.

Therefore, the effectiveness of the log-structured IO optimization on SSD can be summarized as follows:

- The advantage of log-structured IO optimization is still based on the asymmetric performance between sequential IO and random IO. The performance asymmetry is caused by the erase operations instead of the disk head moving. The more time-consuming is the erase operation, the more speedup is obtained by log-structured IO optimization.
- In order to achieve high performance, the implementation of the log-structured IO optimization should not introduce extra IOs for special purpose, such as snapshot.

5. Related Work

5.1 Non-In-Place Update Techniques

Continuous data protection (CDP) [14] [18] is a backup technology automatically saving a copy of every change made to that data to a separate storage location in an enterprise storage system. Another successful example is the Sprite LFS [12], a log-structured file system. The LFS is designed to exploit fast sequential write performance of hard disk, by converting 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 [10]. Shadow paging [13] is a copy-on-write technique for avoiding in-place updates of pages. It needs to modify indexes and block lists when the shadow pages are submitted. This procedure may recurse many times, becoming quite costly.

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 [4] with a log-structured architecture [12]. 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 to flash pages [1]. 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 [6].

File System Most of the file system designs for flash memory are based on Log-structured file system [12], as a way to compensate for the write latency associated with erasures. JFFS, and its successor JFFS2 [3], are journaling file systems for flash. JFFS2 performs wear-leveling with the cleaner selecting a block with valid data at every 100th cleaning, and one with most invalid data at other times. YAFFS [17] is a flash file system for embedded devices.

Database System Previous design for database system on flash memory mainly focused on the embedded systems or sensor networks in a log-structured behavior. FlashDB [9] is a self-tuning database system optimized for sensor networks, with two modes: disk mode for infrequent write, much like regular B^+–tree; log mode for frequent write, employed a log-structured approach. LGeDBMS [5], is a relational database system for mobile phone. For enterprise database design on flash memory, In-Page Logging [8] is proposed. The key idea is to co-locate a data page and its log records in the same physical location.

6. Conclusion and Future Work

Many IO optimization techniques can be applied to the flash-based transaction processing systems. Different IO optimizations with special characteristics of the flash SSD, can yield different transaction throughput. We provided a trace-based analysis at different layers of the kernel through the IO path on SSDs with two different IO optimization techniques at file system level.

As for the future work, we plan to boost the performance of the whole system by combining the characteristics of database system, the IO optimization techniques and the features of flash SSD.

References

[1] Ban, A.: Flash file system. US Patent No. 5404485 (April 1995)

[2] EMC: Specification Sheet: EMC Symmetrix V-Max Storage System. `http://www.emc.com/products/detail/hardware/symmetrix-v-max.htm`

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

[4] Kawaguchi, A., Nishioka, S., Motoda, H.: A Flash-Memory Based File System. In: USENIX Winter. (1995) 155-164

[5] Kim, G.J., Baek, S.C., Lee, H.S., Lee, H.D., Joe, M.J.: LGeDBMS: A Small DBMS for Embedded System with Flash Memory. In: VLDB. (2006) 1255-1258

[6] Kim, J., Kim, J.M., Noh, S.H., Min, S.L., Cho, Y.: A space-efficient flash translation layer for CompactFlash systems. IEEE_J_CE 48(2) (May 2002) 366-375

[7] Konishi, R., Amagai, Y., Sato, K., Hifumi, H., Kihara, S., Moriai, S.: The Linux implementation of a log-structured file system. Operating Systems Review 40(3)(2006) 102-107

[8] Lee, S.W., Moon, B.: Design of flash-based DBMS: an in-page logging approach. In: SIGMOD Conference. (2007) 55-66

[9] Nath, S., Kansal, A.: FlashDB: dynamic self-tuning database for NAND flash. In: IPSN. (2007) 410-419

[10] Neefe, J.M., Roselli, D.S., Costello, A.M., Wang, R.Y., Anderson, T.E.: Improving the Performance of Log-Structured File Systems with Adaptive Methods. In: SOSP. (1997) 238-251

[11] Oracle: ORACLE EXADATA V2: `http://www.oracle.com/us/products/database/exadata/index.htm`

[12] Rosenblum, M., Ousterhout, J.K.: The Design and Implementation of a Log-Structured File System. ACM Trans. Comput. Syst. 10(1) (1992) 26-52

[13] Shenai, K. In: Introduction to database and knowledge-base systems. World Scientific (1992) page 223

[14] Strunk, J.D., Goodson, G.R., Scheinholtz, M.L., Soules, C.A.N., Ganger, G.R.:Self-Securing Storage: Protecting Data in Compromised Systems. In: OSDI. (2000) 165-180

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

[16] Wang, Y., Goda, K., Kitsuregawa, M.: Evaluating Non-In-Place Update Techniques for Flash-based Transaction Processing Systems. In: DEXA (2009), 777-791

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

[18] Zhu, N., Chiueh, T.: Portable and Efficient Continuous Data Protection for Network File Servers. In: DSN. (2007) 687-697