A Log Buffer-Based Flash Translation Layer Using Fully-Associative Sector Translation

SANG-WON LEE
Sungkyunkwan University
DONG-JOO PARK
Soongsil University
TAE-SUN CHUNG
Ajou University
DONG-HO LEE
Hanyang University
SANGWON PARK
Hankook University of Foreign Studies
and
HA-JOO SONG
Pukyong National University

Flash memory is being rapidly deployed as data storage for mobile devices such as PDAs, MP3 players, mobile phones, and digital cameras, mainly because of its low electronic power, nonvolatile storage, high performance, physical stability, and portability. One disadvantage of flash memory is that prewritten data cannot be dynamically overwritten. Before overwriting prewritten data, a time-consuming erase operation on the used blocks must precede, which significantly degrades

This work was supported in part by MIC & IITA through IT Leading R&D Support Project, in part by MIC & IITA through Oversea Post-Doctoral Support Program 2005, in part by the Ministry of Information and Communication, Korea under the ITRC support program supervised by the Institute of Information Technology Assessment, IITA-2005-(C1090-0501-0019), and also supported in part by Seoul R&D Program(10660).
Authors' addresses: Sang-Won Lee, School of Information and Communications Engineering, Sungkyunkwan University, Suwon 440-746, Korea; email: swlee@acm.org; Dong-Joo Park, School of Computing, Soongsil University, Seoul 156-743, Korea; email: djpark@ssu.ac.kr; Tae-Sun Chung, College of Information Technology, Ajou University, Suwon 443-749, Korea; email: tschung@ajou.ac.kr; Dong-Ho Lee, Department of Computer Science and Engineering, Hanyang University, Ansan 426-791, Korea; email: dhlee72@cse.hanyang.ac.kr; Sangwon Park, Information Communication Engineering, Hankook University of Foreign Studies, Yongin 449-791, Korea; email: swpark@hufs.ac.kr; Ha-Joo Song, Division of Electronic, Computer, and Telecommunication, Pukyong National University, Busan 608-737, Korea; email: hajusong@pknu.ac.kr.

the overall write performance of flash memory. In order to solve this "erase-before-write" problem, the flash memory controller can be integrated with a software module, called "flash translation layer (FTL)." Among many FTL schemes available, the log block buffer scheme is considered to be optimum. With this scheme, a small number of log blocks, a kind of write buffer, can improve the performance of write operations by reducing the number of erase operations. However, this scheme can suffer from low space utilization of log blocks. In this paper, we show that there is much room for performance improvement in the log buffer block scheme, and propose an enhanced log block buffer scheme, called FAST (full associative sector translation). Our FAST scheme improves the space utilization of log blocks using fully-associative sector translations for the log block sectors. We also show empirically that our FAST scheme outperforms the pure log block buffer scheme.

ACM Reference Format:

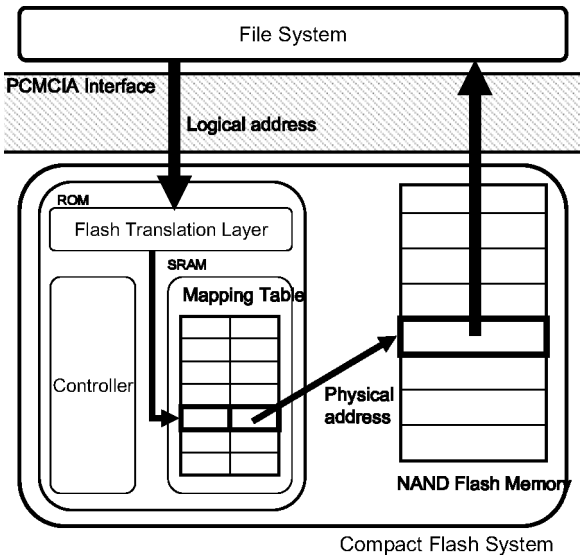
Lee, S.-W., Park, D.-J., Chung, T.-S., Lee, D.-H., Park, S., and Song, H.-J. 2007. A log buffer-based flash translation layer using fully-associative sector translation. ACM Trans. Embedd. Comput. Syst. 6, 3, Article 18 (July 2007), 27 pages. DOI = 10.1145/1275986.1275990 http://doi.acm.org/ 10.1145/1275986.1275990

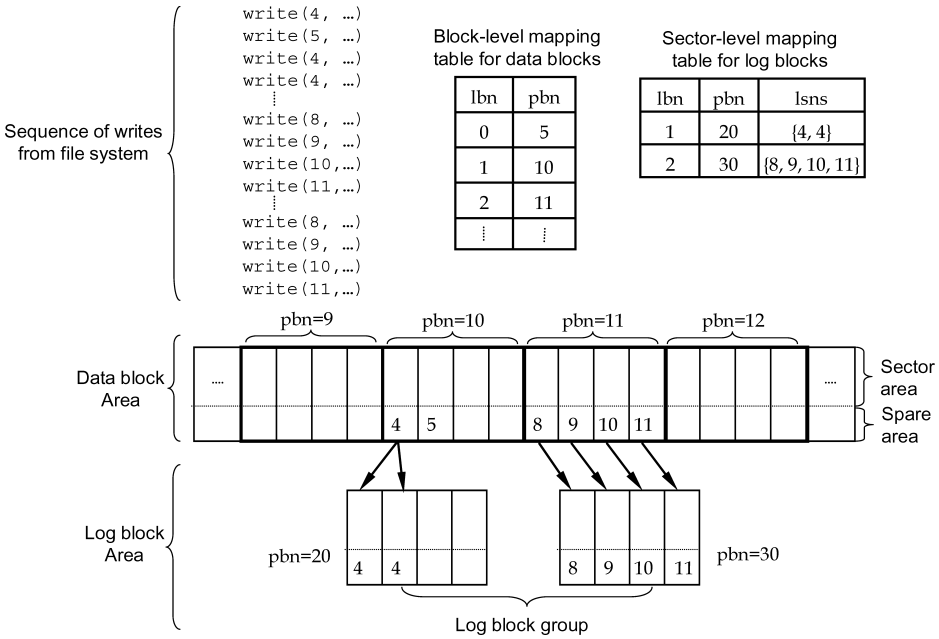
1. INTRODUCTION

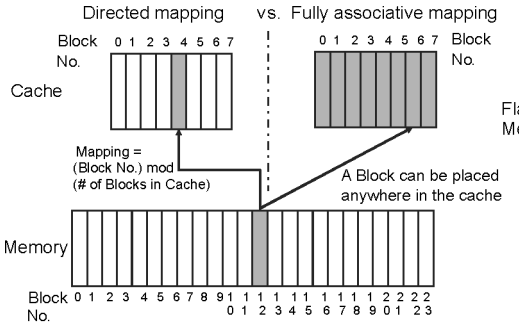
Flash memory is being rapidly deployed as data storage for mobile devices, such as PDAs, MP3 players, mobile phones, and digital cameras, mainly because of its small size, low power consumption, shock resistance, and nonvolatile memory [Douglis et al. 1994; Kim et al. 2002; Gal and Toledo 2005]. Compared to a hard disk with the inevitable mechanical delay in accessing data (that is, seek time and rotational latency), flash memory provides fast uniform random access. For example, the read and write time per sector (typically, 512 bytes) for NAND-flash memory is 15 and 200 μs [Samsung Electronics 2005], respectively, while each operation in contemporary hard disks takes around 10 ms [Hennessy and Patterson 2003].

However, flash memory suffers from the write bandwidth problem. A write operation is slower than a read operation by an order of magnitude. In addition, a write operation may have to be preceded by a costly erase operation because flash memory does not allow overwrites. Unfortunately, write operations are performed in a sector unit, while erase operations are executed in a block unit: usually, one block consists of 32 sectors. An erase operation is very time-consuming, compared to a write operation; usually, a per-block erase time is 2 ms [Samsung Electronics 2005]. These inherent characteristics of flash memory reduce write bandwidth, which is the performance bottleneck in flash-based mobile devices.

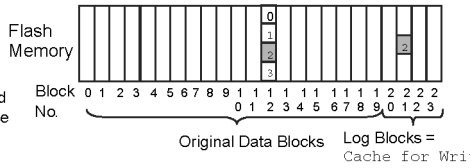
To relieve this performance bottleneck, it is very important to reduce the number of erase operations resulting from write operations. For this, flash memory vendors have adopted an intermediate software module, called flash translation layer (FTL), between the host applications (in general, file systems) and flash memory [Estakhri and Iman 1999; Kim and Lee 2002; Kim et al. 2002;


File System
PCMCIA Interface
Logical address
ROM
Flash Translation Layer
SRAM
Mapping Table
Controller
Physical address
NAND Flash Memory
Compact Flash System

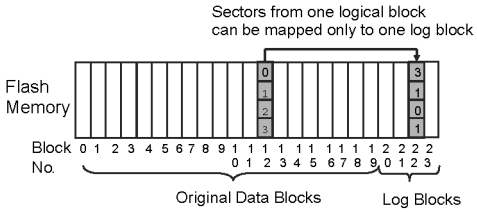

Sequence of writes from file system
write(4, …)
write(5, …)
write(4, …)
write(4, …)
write(8, …)
write(9, …)
write(10,…)
write(11,…)
write(8, …)
write(9, …)
write(10,…)
write(11,…)
Block-level mapping table for data blocks
lbn
pbn
0
5
1
10
2
11
Sector-level mapping table for log blocks
lbn
pbn
lsns
1
20
{4, 4}
2
30
{8, 9, 10, 11}
Data block Area
pbn=9
pbn=10
pbn=11
pbn=12
4
5
8
9
10
11
Sector area
Spare area
Log block Area
pbn=20
4
4
8
9
10
11
pbn=30
Log block group



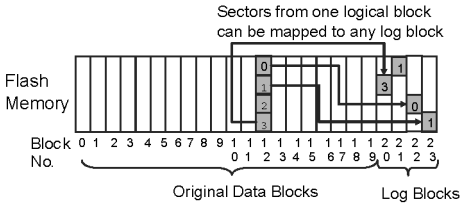
(a) Associativity between memory and CPU cache



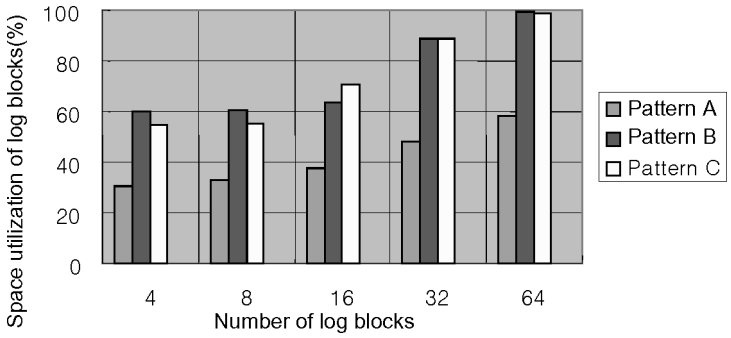
(b) Log blocks: a cache for write operations

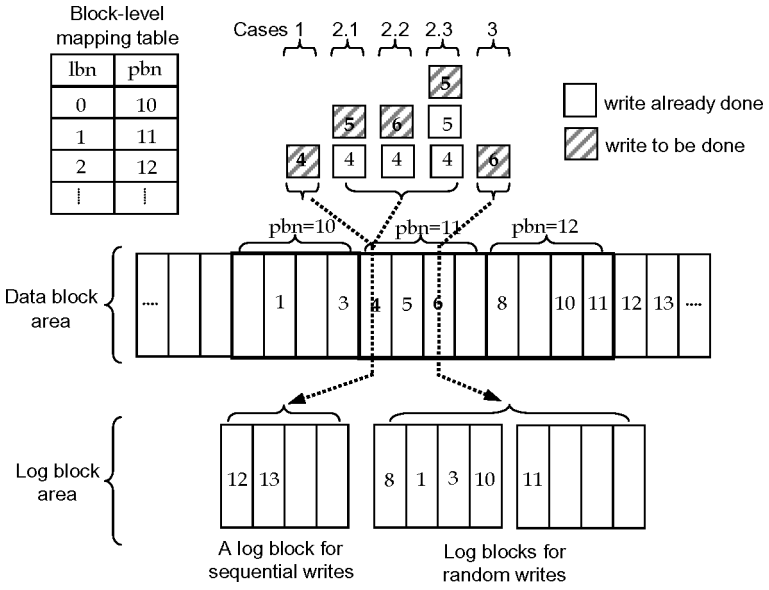


(c) Block associativity in log block scheme



(d) Full associativity between logical sectors and log blocks


Space utilization of log blocks(%)
100
80
60
40
20
0
4
8
16
32
64
Number of log blocks
Pattern A
Pattern B
Pattern C

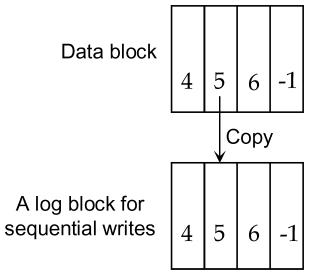

Block-level mapping table
lbn
pbn
0
10
1
11
2
12
Cases 1
2.1
2.2
2.3
3
write already done
write to be done
pbn=10
pbn=11
pbn=12
Data block area
Log block area
A log block for sequential writes
Log blocks for random writes

Algorithm 1 `write`(*lsn*, *data*) /* data are logically written to the sector of lsn */

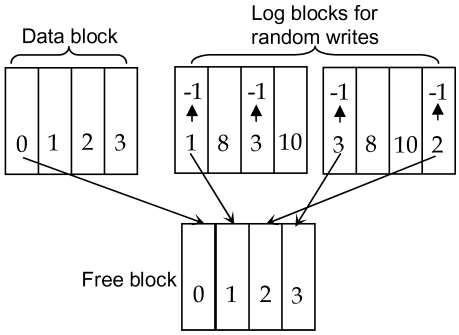
```
1   lbn := lsn div SectorsPerBlock;
2   offset := lsn mod SectorsPerBlock;
3   pbn := getPbnFromBMT(lbn);      /* get pbn from block-level mapping table */
4   if a collision occurs at offset of the data block of pbn
5       call writeToLogblock(lsn, lbn, offset, data);
6   else
7       write data at offset in the data block of pbn;
8   end if
```

Algorithm 2 `writeToLogblock`(*lsn, lbn, offset, data*)

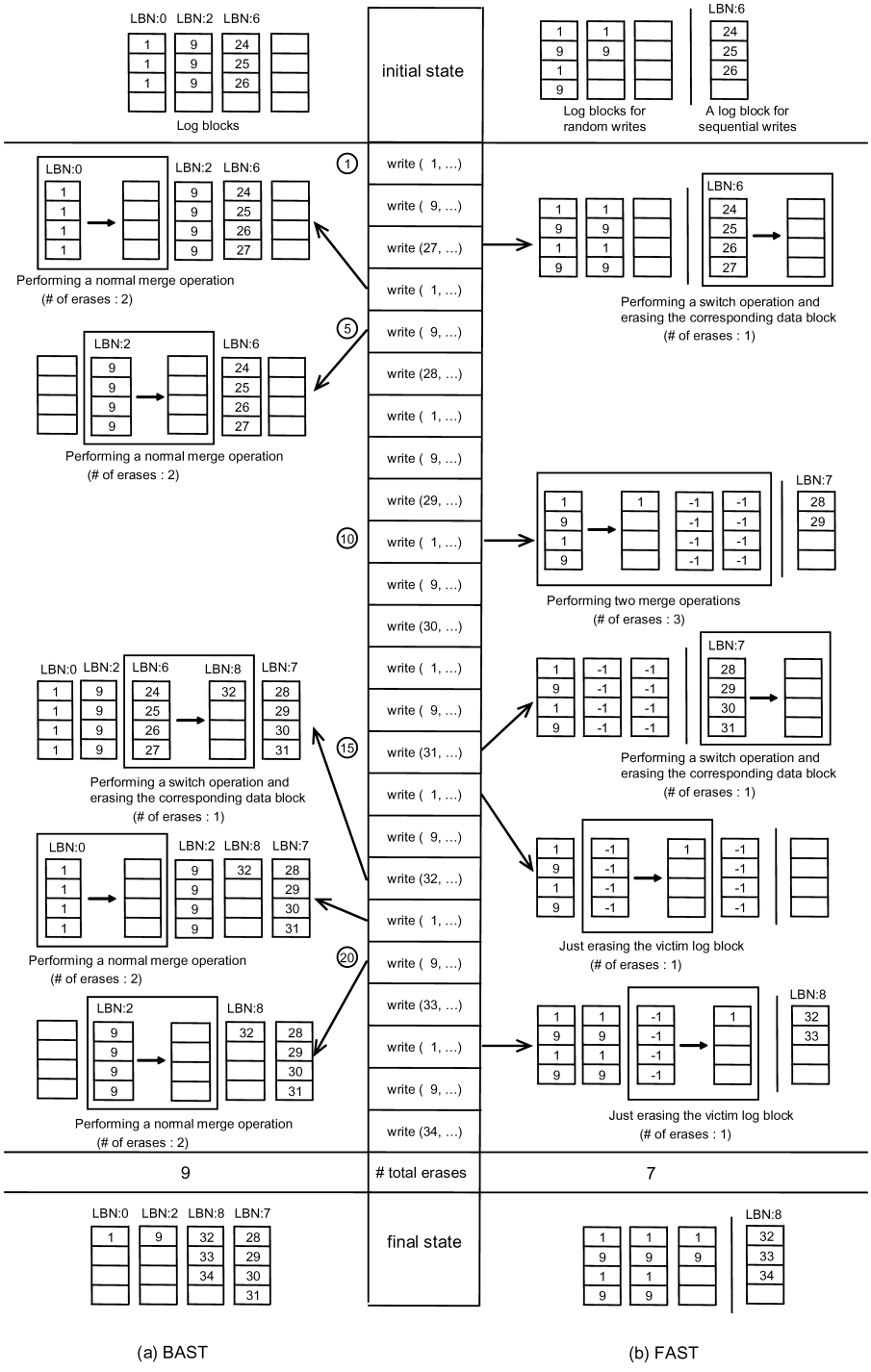
```
1   if offset is zero                          /* Case 1 in Figure 5 */
2       if there are no empty sectors in the SW log block
            /* the log block is filled with sequentially written sectors */
3           perform a switch operation between the SW log block and
            its corresponding data block;
            /* after switch, the data block is erased and returned to the free-block list */
4       else
            /* before merge, a new block is allocated from the free-block list */
5           merge the SW log block with its corresponding data block;
            /* after merge, the two blocks are erased and returned to the free-block list */
6       end if
7       get a block from the free-block list and use it as a SW log block;
8       append data to the SW log block;
9       update the SW log block part of the sector-mapping table;
10  else
11      if the current owner of the SW log block is the same with lbn
12          last_lsn := getLastlsnFromSMT(lbn);        /* SMT: sector-mapping table */
13          if lsn is equivalent with (last_lsn+1)     /* Case 2.1 in Figure 5 */
14              append data to the SW log block;
15          else                                       /* Case 2.2 and 2.3 in Figure 5*/
16              merge the SW log block with its corresponding data block;
17              get a block from the free-block list and use it as a SW log block;
18          end if
19          update the SW log block part of the sector-mapping table;
20      else                                           /* Case 3 in Figure 5 */
21          if there are no rooms in the RW log blocks to write data
22              select the first block of the RW log block list as a victim;
23              merge the victim with its corresponding data block;
24              get a block from the free-block list and add it to the end of
                the RW log block list;
25              update the RW log block part of the sector-mapping table;
26          end if
27          append data to the RW log blocks;
28      end if
29  end if
```

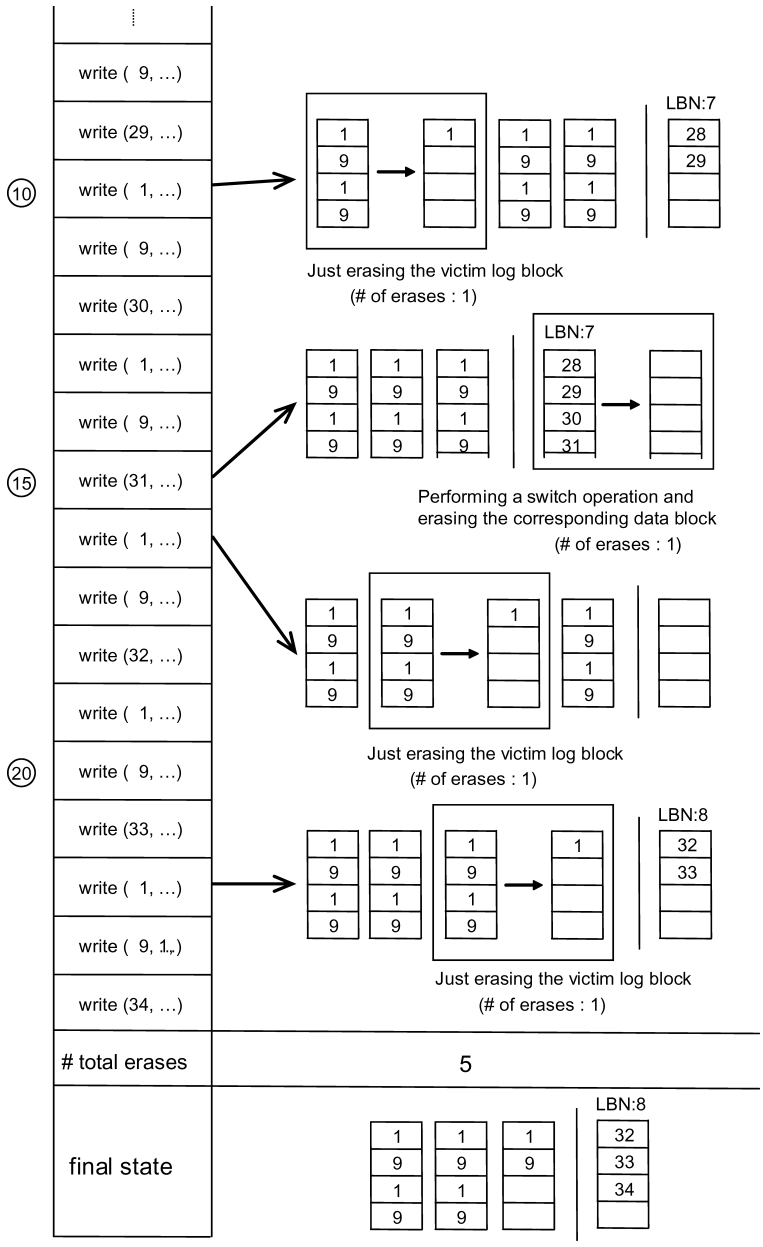


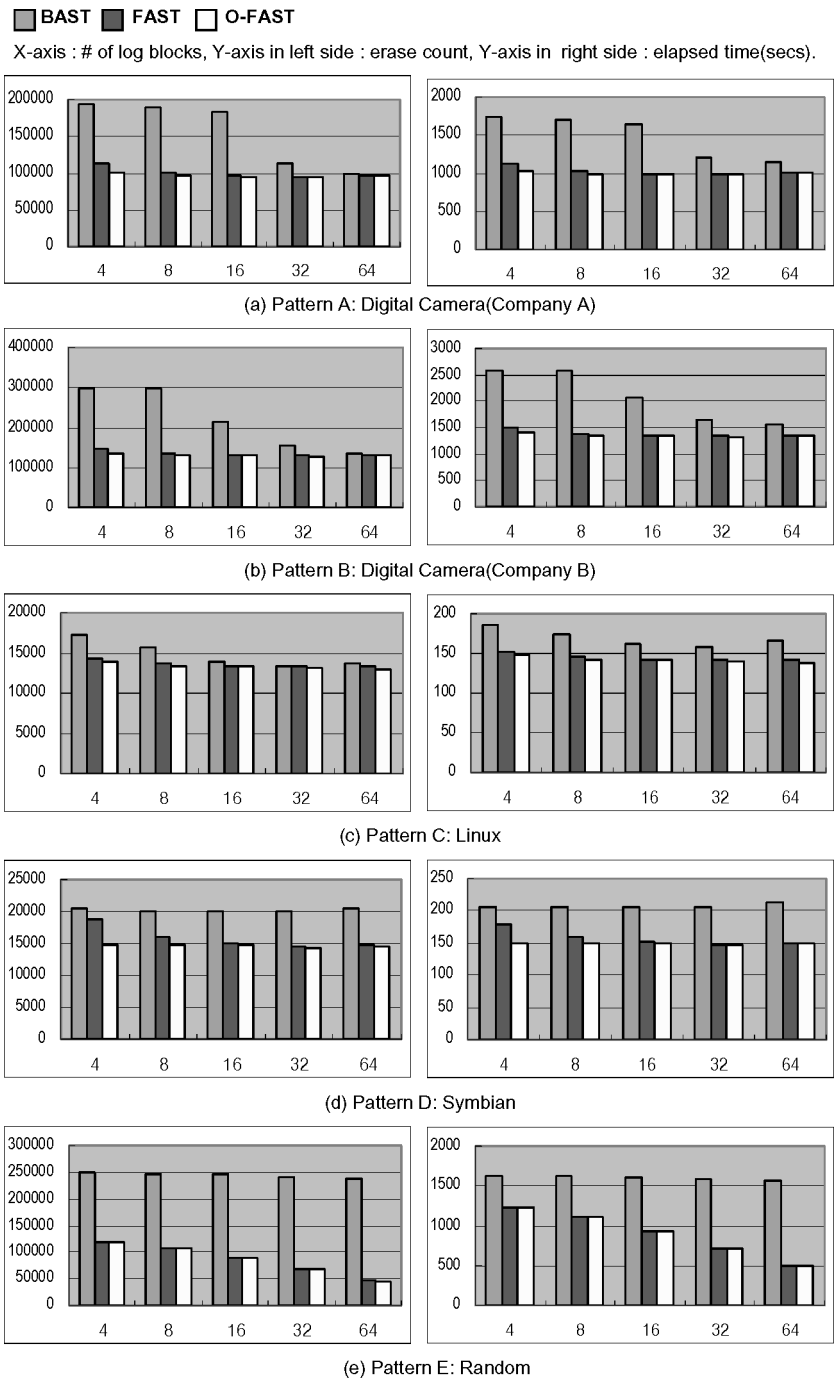
(a) A merge in the SW log block



(b) A merge in the RW log blocks


initial state
Log blocks
Log blocks for random writes
A log block for sequential writes
write (1, …)
write (9, …)
write (27, …)
write (1, …)
write (9, …)
write (28, …)
write (1, …)
write (9, …)
write (29, …)
write (1, …)
write (9, …)
write (30, …)
write (1, …)
write (9, …)
write (31, …)
write (1, …)
write (9, …)
write (32, …)
write (1, …)
write (9, …)
write (33, …)
write (1, …)
write (9, …)
write (34, …)
Performing a normal merge operation (# of erases : 2)
Performing a switch operation and erasing the corresponding data block (# of erases : 1)
Performing a normal merge operation (# of erases : 2)
Performing two merge operations (# of erases : 3)
Performing a switch operation and erasing the corresponding data block (# of erases : 1)
Performing a switch operation and erasing the corresponding data block (# of erases : 1)
Just erasing the victim log block (# of erases : 1)
Performing a normal merge operation (# of erases : 2)
Just erasing the victim log block (# of erases : 1)
Performing a normal merge operation (# of erases : 2)
9
total erases
7
final state
(a) BAST
(b) FAST


write (9, …)
write (29, …)
write (1, …)
write (9, …)
write (30, …)
write (1, …)
write (9, …)
write (31, …)
write (1, …)
write (9, …)
write (32, …)
write (1, …)
write (9, …)
write (33, …)
write (1, …)
write (9, 1, …)
write (34, …)
10
15
20
LBN:7
Just erasing the victim log block
(# of erases : 1)
LBN:7
Performing a switch operation and
erasing the corresponding data block
(# of erases : 1)
Just erasing the victim log block
(# of erases : 1)
LBN:8
Just erasing the victim log block
(# of erases : 1)
total erases
5
final state
LBN:8


BAST FAST O-FAST
X-axis : # of log blocks, Y-axis in left side : erase count, Y-axis in right side : elapsed time(secs).
200000
150000
100000
50000
0
2000
1500
1000
500
4 8 16 32 64
(a) Pattern A: Digital Camera(Company A)
400000
300000
200000
100000
3000
2500
2000
1500
(b) Pattern B: Digital Camera(Company B)
20000
15000
10000
5000
200
150
100
50
(c) Pattern C: Linux
25000
250
(d) Pattern D: Symbian
300000
250000
(e) Pattern E: Random