JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 29, 947-967 (2013)

DOI:10.1688/JISE.2013.29.5.9

Flash-Aware Cost Model for Embedded Database Query Optimizer*

SANGWON PARK
Information and Communication Engineering
Hankuk University of Foreign Studies
Gyeonggido, 447-791 Korea

Flash memory is suitable for embedded devices because it offers small, non-volatile, impact-resistant and low power consumption. However, in flash memory, the speeds of the read, write, and erase operations are different. In addition, flash memory features hardware characteristics including erase-before-write. Therefore, a flash translation layer (FTL) is required to efficiently perform operations to the flash memory. FTL allows the file system to consider the flash memory as a block device, and the existing file system can be used without any additional modifications. Most databases use the disk-aware cost model to perform query optimization. If the storage device for the database is replaced by flash memory, the cost model for the database query optimization must be changed to the flash-aware cost model. In this study, we propose a cost model for flash memory, and we compare the differences between the flash-aware and disk-aware cost models.

Keywords: flash-aware cost model, database, query optimization, flash memory, flash translation layer

1. INTRODUCTION

Several recently developed mobile devices, such as digital cameras, MP3 players, and mobile phones use flash memory as their medium of storage. Flash memory is a storage device that offers the advantages of small size, large storage capacity, low power consumption, non-volatile memory, high access speed and high impact-resistance over conventional hard disks. However, unlike hard disks that can directly perform overwrite operations, flash memory is characterized by erase-before-write operation. When a sector in flash memory is overwritten, the flash memory initializes the block to which a sector belongs by erasing the block, and subsequently it performs write operations to the memory. Moreover, the processing speeds of the read, write, and erase operations are different. The erase operations are much slower than the read and write operations [1]. In addition, frequent erase operations degrade flash memory performance. Table 1 compares the speeds of the read, write, and erase operations of the flash memory and the hard disk.

A flash translation layer (FTL) [2-8] is used to improve flash memory performance. When a write operation is performed in a sector, the FTL uses a mapping table to record the new data to the appropriate sector at another physical location; the old block is erased later when required. This reduces erase operations that are the causes of the flash memory performance degradation. In addition, FTL allows the flash memory to be considered as a block device by the operating system. Therefore, the flash memory can be used together with the existing file system.

Received May 16, 2011; revised July 20 & September 4, 2011; accepted November 2, 2011.
Communicated by Tei-Wei Kuo.
* This work was supported by Hankuk University of Foreign Studies Research Fund of 2011.

Table 1. Parameters of the disk and the flash memory used in this study.

Devices	Model	Operation Time		
		Read	Write	Erase
Disk	Seagate Barracuda 7200.7 ST380011A	12.7ms (2KB)	13.7ms (2KB)	N/A
Flash Memory	Samsung K9WAG08U1A 16Gbits SLC NAND	80μs (2KB)	200μs (2KB)	1.5ms (128KB)

In order to calculate the query processing cost for conventional disk-aware database, the number of disk accesses is the most important factor. However, there are large differences between the read, write, and erase operation times in flash memories, a new cost model is required for flash-aware databases. Recently introduced solid state drives (SSD) have similar read and write operation times; this is because an SSD has many internal chips functioning with parallel operations. However, such expensive systems cannot be applied to embedded systems. Therefore, the characteristics of the flash memory must be considered using a new cost model.

Basically, the disk-aware cost model assumes that reads and writes have the same operation cost. However, in flash memory, write operations are very slow compared with read operations. However, due to the erase-before-write architecture characteristics of flash memory, FTL has to be used for reads and writes. FTL replaces read and write operations of the file system with read, write, and erase operations on flash memory. Therefore, since file system's reads and writes cause additional operations, accurate cost estimation is not possible simply by using the number of read and write operations as in the disk-aware cost model. Thus, a flash-aware cost model is required.

In this study, we proposed a new cost model for a join query in the flash memory; the join query is considered the most expensive query operation. This model was tested in five FTLs. Subsequently, we analyzed the cost models for each join query from the test results.

This paper is organized as follows: Section 2 describes the flash memory and the FTL system. Section 3 describes parameters α and β, which were used to calculate the cost of operations to the flash memory. Section 4 uses α and β to propose a new model by modifying the disk-based cost model to the flash-aware cost model. Section 5 describes the design of the query processing simulation system. In section 6, the join costs that are calculated using the cost models are compared for five FTLs, and the FTL simulator is used to compare the cost measured in the virtual flash memory with the cost measured in the flash-embedded board. Section 7 presents our conclusions and areas of future study.

2. RELATED WORK

Initially, flash memory was mainly used as a storage medium for devices that were required to store infrequently changed data and system codes. However, it is now being used, as a storage medium for files due to its advantages such as high reliability and low

power consumption. Flash memory has different operation times for read, write, and erase operations. In the case of a large block, the read and write operations are performed in a 2KB page unit, and the erase operation is performed in a 128KB block unit [1]. In the case of a small block, the read and write operations are performed in a 512 bytes page unit, and the erase operation is performed in a 32KB block unit. When data are overwritten to a page in the flash memory, the block to which the page belongs must first be erased. For example, if all pages in a block have data, the overwrite operation on a page requires all the valid pages in the block to be moved to another free block. After the block is erased, the original page must be moved back. In the worst case, 126 read operations, 126 write operations, and one erase operation are required to move 63 valid pages in order to write to a page in the case of a large block. In addition, because each block has an erasing limit of 100,000 to 1,000,000 times, it is required to implement the wear-leveling technique [9, 10].

An FTL is the system software required to solve these problems. The FTL is present between the flash memory and the file system. The read and write operations of the file system are delivered to the FTL, which reduces the read, write, and erase operations in the flash memory via efficient mapping to rapidly process the requests of the file system.

The FTL algorithm uses sector, block, and hybrid mapping methods to map the flash memory [11]. Sector mapping [2] creates a mapping table by sector, and maps the logical sector number of the file system to the physical sector number of the flash memory. Block mapping [4, 5, 13] reduces the mapping table size by mapping in block units. Sector mapping enables very fast operations because the physical sector number of the flash memory can be identified with a single mapping table search. Block mapping uses a RAM with a smaller capacity when compared with that for sector mapping because the number of blocks in the flash memory defines the size of the mapping information. However, the overwrite operation cannot be efficiently performed because the sector location does not change in the block. Hybrid mapping [3, 6, 12] reduces memory usage for recording the mapping table by using block mapping. In addition, this mapping method performs sector mapping within a block, thereby using the advantages of both block and sector mapping methods.

There are two ways to store data at a sector in a block in the flash memory. Recording to a sector in a fixed location in a block is called the in-place method, and storing to a sector in a variable location is called the out-of-place method [11].

Kim *et al.* [6] used the hybrid mapping algorithm. In this hybrid mapping method, for a given number of sectors in a block, the sector numbers are recorded in each sector's spare area. Thus, although this algorithm involves more read operations, it requires fewer erase and write operations. In another study, Estakhri *et al.* [12] used block mapping and recorded a sector using the in-place method. In this method, when an overwrite occurs, the algorithm reduces the number of erase operations by recording the relevant data to a replacement block. In this study, five algorithms were selected from many FTL algorithms that use typical mapping and recording methods.

File system uses logical sector number to access storages, and FTL translate this logical sector number to physical sector number. A block is composed of several sectors. In Mitsubishi algorithm [5], one logical block is mapped to one physical block. A physical block in Mitsubishi algorithm is composed of a general sector area and a space sector area. If a logical block consists of m sectors, a physical block consists of m general sec-

tors and some additional *n* space sectors. In-place mapping is used to find a sector in the general sector area, and out-of-place mapping is used to find the sector in the space sector area. The space sectors are used as buffers in order to reduce the number of erase operations when overwrite occurs.

In FMAX algorithm [4], one logical block is mapped to the two physical blocks known as a primary block and a replacement block. In-place mapping is used at the primary block, and out-of-place mapping method is used at the replacement block. The replacement block is used as buffer in order to reduce the number of erase operations when overwrite occurs.

In log block algorithm [6], it assumes that the operations are composed of numerous long sequential writes and a small number of random overwrite operations. The blocks in the log block algorithm are composed of numerous data blocks and several log blocks. In-place mapping is used at data blocks; and out-of-place mapping is used at log blocks in order to reduce the number of erase operations when random overwrite operations are occurred. FAST [7] classifies log blocks as a sequential log block and random log blocks. The sequential log block is used for sequential write, and random log blocks are used for random overwrites.

In superblock algorithm [8], a set of adjacent logical blocks shares data blocks and update blocks. The logical pages within a superblock are allowed to be freely located in one of the allocated physical blocks for the superblock.

Table 2 shows the comparison of the selected algorithm which include the FMAX of M-System [4], algorithm of Mitsubishi (MITS) [5], LOG (with log blocks) [6], FAST [7], and superblock [9] systems.

Table 2. Comparison of selected FTL algorithms.

FTL	Mapping	Overwrite	Features
FMAX	block	Records to the data block in with the in-place method and to the replacement block with the out-of-place method	Every data block has a corresponding replacement block.
MITS	block	Records to the spare sector with the out-of-place method	Spare sector in the block is used.
Superblock	hybrid	Allow logical pages within a superblock to be freely located in one of the allocated physical blocks	Several superblock are used for overwrite.
LOG	hybrid	Records to the log block with the out-of-place method and to the data block with the in-place method	One log block is assigned to one data block. There are several log blocks.
FAST	hybrid	Records to the random log block with the out-of-place method and to the sequential log block with in-place method	One log block for sequential overwrite and several log blocks for random overwrite.

3. AVERAGE COSTS OF READ AND WRITE OPERATIONS TO THE FLASH MEMORY

In the disk-aware cost model, the read and write operations require the same time and their operational costs are identical. Accordingly, the query processing cost in this case is defined as the number of I/O operations. In contrast, flash memory involves three

operations – read, write, and erase – and they require different operation times. In addition, because the read and write requests are performed via mapping in FTL, additional read, write, and erase operations other than those requested occur in the flash memory. Thus, one read or one write request produces multiple read, write, and erase operations. Therefore, the additional cost of one read or one write request can be calculated for each FTL. In FTL, the ratio of the additional cost that arises from the write request is denoted as α, and the ratio of the additional cost that arises from the read request is denoted as β. Table 3 lists the parameters of the equations.

Table 3. Parameters of the cost model.

C_r	time to read a page from the flash memory (μs)
C_w	time to write a page to the flash memory (μs)
C_e	time to erase a block in the flash memory (μs)
m	the number of buses in a NAND flash controller
S_p	size of a block of database (KB)
S_{fp}	size of a page in the NAND flash (KB)
n_{dr}, n_{dw}	number of read and write operations to the disk in the database system
n_{rr}, n_{rw}, n_{re}	number of additional read, write, and erase operations to the flash memory when a read operation occurred in database system
n_{wr}, n_{ww}, n_{we}	number of additional read, write, and erase operations to the flash memory when a write operation occurred in database system

The equations for cost calculations involving a write request are as follows,

$$k = \frac{S_p}{S_{fp} m}, \tag{1}$$

$$p_w = kC_w, \tag{2}$$

$$P_w = \sum_{i=1}^{n} p_{w_i} = k \cdot n_{dw} \cdot C_w, \tag{3}$$

$$P'_w = n_{wr}C_r + n_{ww}C_w + n_{we}C_e, \tag{4}$$

$$\alpha = \frac{P'_w}{P_w} = \frac{n_{wr}C_r + n_{ww}C_w + n_{we}C_e}{n_{dw} \cdot p_w} = \frac{n_{wr}C_r + n_{ww}C_w + n_{we}C_e}{k \cdot n_{dw} \cdot C_w}. \tag{5}$$

Writing to one block in the database results in the writing of k pages in the flash memory. Eq. (1) is used to calculate k for a database block size S_p (KB), with a flash memory page size of S_{fp} (generally 2KB in the case of large block), and the number of chip interleaving is denoted as m. Chip interleaving is a method of sending a single command simultaneously to m flash memory chips through m buses. If the chip interleaving value is 1, one command is transferred to a chip; and if it is 2, the command is simultaneously transferred to two chips. For example, if the size of a page in the database (S_p) is 8KB and the chip interleaving (m) is 1, k is calculated as 8KB / (2KB × 1) = 4. That is, an 8KB write request produces a write operation that uses four pages in the flash memory.

The term p_w in Eq. (2) denotes the cost that is required for recording to a block in the database. The cost is calculated by multiplying the number of write operations to the

pages for a write request for a block (k) with the cost of the write operation to a page in the flash memory (C_w). The term P_w in Eq. (3) denotes the sum of write costs of the flash memory arising from the n_{dw} write requests from the file system; thus, P_w denotes the optimal write cost. The term P'_w in Eq. (4) is the sum of the numbers of read operations (n_{wr}), write operations (n_{ww}), and erase operations (n_{we}) that occur upon the write request of the file system. The ratio of the additional cost that arises from the write request, α, is calculated as in Eq. (5) by dividing the flash memory cost that arises from the write request (P'_w) by the optimal write cost (P_w); this ratio is the average ratio of the additional cost that arises from a write request to a block in the database.

For example, if the numbers of read, write, and erase operations that arises from a write request for 100 blocks in the database are 200, 750, and 50, respectively, α is calculated using Eq. (5), as follows:

$$[(200/100) \times 80 + (750/100) \times 200 + (50/100) \times 1{,}500] / (4 \times 200) = 4.7875.$$

It means that additional cost of write request from file system is 4.7875 times compared to optimal cost.

The equations for cost calculations involving a read request are as follows:

$$p_r = kC_r, \tag{6}$$

$$P_r = \sum_{i=1}^{n} p_{r_i} = k \cdot n_{dr} \cdot C_r \tag{7}$$

$$P_r^{'} = n_{rr}C_r + n_{rw}C_r + n_{re}C_e = n_{rr}C_r \qquad \because n_{rw}, n_{re} = 0 \tag{8}$$

$$\beta = \frac{P_r^{'}}{P_r} = \frac{n_{rr}C_r}{n_{dr} \cdot p_r} = \frac{n_{rr}}{k \cdot n_{dr}} \tag{9}$$

The additional cost that arises from the read request (β) can be calculated in a similar manner. The term p_r in Eq. (6) denotes the time spent in the flash memory to read a block in the database using the k value calculated in Eq. (1). The term P_r in Eq. (7) is the sum of the read operation costs when the number of read requests from file system is n_{dr}. The term P'_r in Eq. (8) denotes the sum of the read, write, and erase costs that are incurred upon a read request sent to the flash memory. The terms n_{rr}, n_{rw} and n_{re} indicate the accumulated numbers of read, write, and erase operations, respectively, for a received read request. Because the write and erase operations are not performed for a read request, only the term n_{rr} is not zero. Accordingly, P'_r denotes the cost of the read and additional read operations (n_{rr}) that arise from the read request (n_{dr}). The terms α and β are used to estimate the additional cost that arises from read and write operations to the flash memory. In this study, these terms are used to mainly estimate the additional cost in the flash-aware cost model.

To obtain the α and β values in FTL, additional operations have to be defined when creating FTL. Linux is frequently used as an embedded operating system. To install FTL on Linux, a device driver has to be made. Linux supports proc file system that provides a method to obtain values inside the kernel in user mode. That is, a file that gets the α and β values is created on the proc file system when creating the device driver using FTL and values are read from that file.

4. FLASH-AWARE JOIN COST MODEL

Among the many relational database operations, the cost model of the join operation exerts the most influence on query optimization. The join operation accesses multiple relations when performing queries; therefore, it has a much higher cost than that of other operations. Therefore, in this study, we apply the flash-aware cost model only to the join operation. In order to formulate the flash-aware cost model, we defined the following parameters:

C_{disk}: Cost of the disk
C_{flash}: Cost of the flash memory
M: Number of buffer blocks
b_r, b_s: Number of buffer blocks of r and s relations
n_r, n_s: Number of records of r and s relations

In this study, we present a flash-aware cost model for the following four representative join types: the block nested loop join, merge join, hash join and indexed nested loop join [14]. While the disk-aware cost model estimates the results based on the number of I/Os, the flash-aware cost model estimates the results based on the required operation time. Therefore, the flash-aware cost model was defined by classifying the disk-aware I/O costs as write and read costs, and multiplying them by α and β, which are the ratios of the additional costs for the write and read operations, respectively.

4.1 Block Nested Loop Join (BNLJ)

The cost equations for the block nested loop join are given as follows:

$$C_{disk}(BNLJ) = b_r \cdot b_s + b_r \quad \text{if } b_r \text{ or } b_s < M \text{ then } C_{disk}(BNLJ) = b_r + b_s, \tag{10}$$

$$C_{flash}(BNLJ) = k\beta C_r(b_r \cdot b_s + b_r) \quad \text{if } b_r \text{ or } b_s < M \text{ then } C_{flash}(BNLJ) = k\beta C_r(b_r + b_s). \tag{11}$$

In the disk-aware cost model in Eq. (10), relation r is called the outer relation and relation s the inner relation of the join, since the loop for r encloses the loop for s. Every block of the inner relation is paired with every block of the outer relation. Within each pair of blocks, every tuple in one block is paired with every tuple in the outer block, to generate all pairs of tuples [14]. Therefore, when the two relations cannot be loaded on the memory at the same time, $b_s \cdot b_r$ repetitions are required; and since b_r itself has to be read, $b_r \cdot b_s + b_r$ block accesses are required. In the best case when the two relations can be loaded to the memory at the same time, there will be $b_r + b_s$ block accesses.

The time required by a flash memory to read a block is kbC_r. The k is the number of pages in a block in the flash memory, C_r is the cost of reading a page of the flash memory, and β is the additional cost for reading flash memory. Therefore, the cost of a flash memory of BNLJ is determined by multiplying $b_r \cdot b_s + b_r$ and kbC_r.

4.2 Merge Join (MJ)

The cost equations for the merge join are given as follows,

$$C_{disk}(MJ) = b_r + b_s + 2b_r(\left\lceil \log_{M-1} \frac{b_r}{M} \right\rceil + 1) + 2b_s(\left\lceil \log_{M-1} \frac{b_s}{M} \right\rceil + 1)$$
$$= 3(b_r + b_s) + 2b_r \left\lceil \log_{M-1} \frac{b_r}{M} \right\rceil + 2b_s \left\lceil \log_{M-1} \frac{b_s}{M} \right\rceil, \quad (12)$$

$$C_{flash}(MJ) = k(\alpha C_w + 2\beta C_r)(b_r + b_s) +$$
$$k(\alpha C_w + \beta C_r)(b_r \left\lceil \log_{M-1} \frac{b_r}{M} \right\rceil + b_s \left\lceil \log_{M-1} \frac{b_s}{M} \right\rceil). \quad (13)$$

If the relations are in sorted order, tuples with the same value on the join attributes are in consecutive order. In that case, each tuple in the sorted relation needs to be read once only. Therefore, each block is also read only once from the disk [14]. If either of the input relations r and s is not sorted on the join attributes, the relations r and s should be sorted and joined in MJ method. If the size of the relation is larger than main memory, merge sort algorithm is used to sort the relations. The join operation is performed with the two sorted relations, and the blocks of the relations are read consecutively after sorting two relations.

The disk-aware cost model (Eq. 12) calculates the cost of merge sorting the relations r and s and the cost of reading the two sorted relations. First, sorted runs are created by splitting the relations into the size of M in order to sort the relations in the memory. The number of initial runs is b_r/M. To sort the runs, each run must be read from the disk, and the records of each run must be sorted in the memory, and then written on a disk. After all the initial runs are sorted, $M - 1$ runs are merged into a run for the next stage. This is recursively repeated until only one run is left, which is $[\log_{M-1}(b_r/M)]$ times. Therefore, merge sorting of relation r requires reading relation r and writing it according to the repetitions of $[\log_{M-1}(b_r/M)]$. Thus, the number of I/O operations becomes $2b_r([\log_{M-1}(b_r/M)] + 1)$. Because relation s goes through the same process, the required number of I/O operations of relation s is $2b_s([\log_{M-1}(b_s/M)] + 1)$. For the join operation, each sorted runs of the relations must be read in order. The cost of this join operation is $b_r + b_s$. Therefore, the total cost is computed with Eq. (12).

In the flash-aware cost model (Eq. 13), the costs of the read and write operations should be separately considered in the calculation. When a run is made, M blocks are read from flash memory at first, and the records in the blocks are sorted in the main memory, and then the sorted blocks are written to the flash memory. In the flash-aware cost model, the write cost is calculated by multiplying the number of write operations by the cost of writing one block ($k\alpha C_w$), and the read cost is calculated by multiplying the number of read operations by the cost of reading one block in the database ($k\beta C_r$).

4.3 Hash Join (HJ)

The cost equations for the hash join are given as follows:

$$C_{disk}(HJ) = b_r + b_s + 2(b_r + b_s)\lceil \log_{M-1} b_s - 1 \rceil \qquad \text{(where } b_r < b_s\text{)} \tag{14}$$

$$C_{flash}(HJ) = k\beta C_r(b_r + b_s) + k(\alpha C_w + \beta C_r)(b_r + b_s)\lceil \log_{M-1}(b_s) - 1 \rceil \qquad \text{(where } b_r < b_s\text{)} \tag{15}$$

In the disk-aware cost model (Eq. 14), the cost is calculated by summing the cost of the hash construction and the cost of reading the constructed hashed partitions. Recursive partitioning is required while making the hash when the relation is larger than the size of main memory M. The recursive partitioning is performed $[\log_{M-1}b_s - 1]$ times until the number of blocks in each partition becomes smaller than M. To partition relation r, each partition must be read into the memory, and must be rewritten on the new divided partition. As the partitioning of relation r requires recursive $[\log_{M-1}b_s - 1]$ repetitions, the total cost of the partitioning becomes $2b_r[\log_{M-1}b_s - 1]$. The multiplication by two is because two I/Os are required to divide a partition; one is a read operation from the old partition and the other is a write operation to the new partition. The total cost of partitioning relation s that was found in this manner is $2b_s[\log_{M-1}b_s - 1]$. After the partitions for relation r and relation s are made, each partition must be read for the join operation. The cost of reading all the partitions is the same as that of the number of blocks of relation r and relation s, which is $b_r + b_s$. The cost of the disk-aware cost model is shown in Eq. (14).

In the flash-aware cost model (Eq. 15), the read and write operations are performed when the hash is constructed. Therefore, the sum of the read cost ($k\beta C_r$) and the write cost ($k\alpha C_w$) are multiplied by the hash construction cost. In addition, the cost of reading the hashing results is multiplied with the read cost ($k\beta C_r$).

4.4 Indexed Nested Loop Join (INLJ)

If there is an index in either of the relations r and s, the indexed nested loop join can be performed. If there is an index in relation s, each record of relation r is joined with relation s by referring to the index.

The cost equations for the indexed nested loop join are given as follows:

$$C_{disk}(INLJ) = b_r + n_r \cdot \lceil \log_{f_B} n_s \rceil, \tag{16}$$

$$C_{flash}(INLJ) = k\beta C_r(b_r + n_r \cdot \lceil \log_{f_B} n_s \rceil). \tag{17}$$

In the disk-aware cost model (Eq. 16), it is assumed that the B+ tree index exists in relation s. The cost is calculated by adding the index searching cost to find the join result for a record in relation r. The term f_B denotes the size of the entry that can be stored in a node of the B+ tree. The height of the B+ tree is $[\log_{f_B}n_s]$, which is the cost of a single selection on s using the join condition. For the joining operation, the index must be searched for each tuple of relation r. Because the number of tuples of relation r is n_r, the cost of index searching is $n_r[\log_{f_B}n_s]$. Thus, the cost in the disk-aware cost model is the sum of the cost of reading relation r, which is b_r, and the cost of index searching.

The flash-aware cost model is defined by multiplying the read cost of the flash mem-

ory for reading a block in the buffer ($k\beta C_r$) with the cost of the disk-aware cost model.

5. QUERY PROCESSING SIMULATION SYSTEM

In this study, a query processing simulator and an FTL simulator were used to measure the query processing costs. A trace executor that can be operated in the embedded board was used to verify the results. As observed in Fig. 1, the query processing simulator consists of a query processor and a buffer manager. The query processor requests the I/O required to process the query for the buffer manager. The buffer manager manages the buffer in the least recently used (LRU) method. It reads and writes the pages in the buffer to the database. The FTL simulator consists of the FTL and the virtual flash memory. In order to verify the accuracy of the results obtained from the FTL simulator, the operation that was transferred from the FTL to the virtual flash memory was performed in the flash-embedded board.

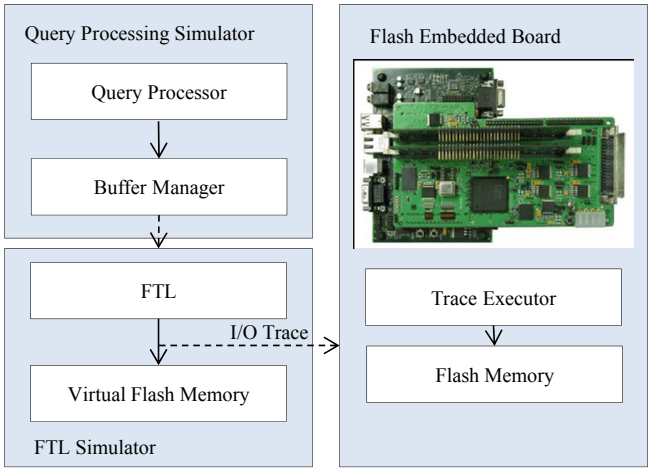


Fig. 1. Diagram of query processing simulation system.

The flash-embedded board consists of the EDB9315A board, daughter board, flash memory and the data acquisition (DAQ) device. The embedded Linux Kernel 2.6.11 was installed on the flash-embedded board, and a system call was added to the kernel to access the flash memory and perform the read, write, and erase operations. The trace executor was used to measure the time spent for the read, write, and erase operations that were produced by the join operation. Using the measured time results, we selected the algorithm that required the least operation time from among the chosen join algorithms; all the algorithms were operated under identical conditions. The results were compared with the cost models.

In order to measure the operation time required for the flash memory, we used a DAQ, as shown in Fig. 2. The DAQ device is a data collection device (provided by National Instruments) that can be connected to the flash-embedded board to measure time and power consumption.

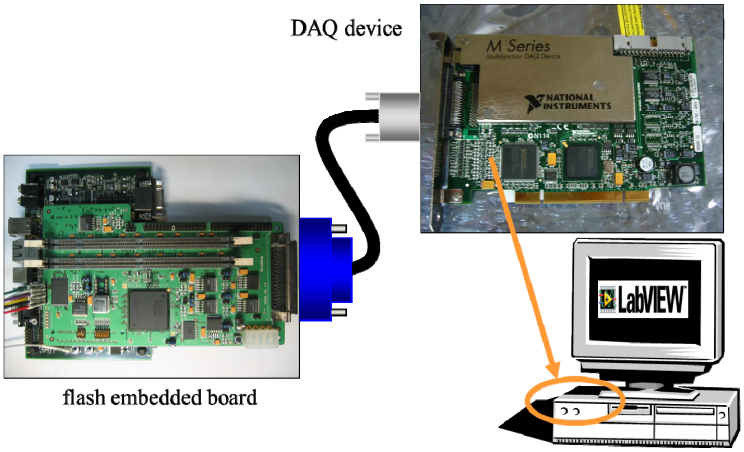


Fig. 2. Experiment system architecture.

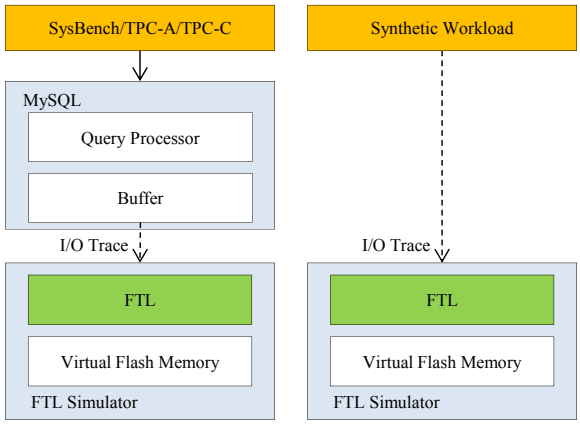


Fig. 3. Architecture for determining α and β using workloads.

6. EXPERIMENT RESULTS

As shown in Fig. 3, the parameters α and β, which are required for the flash-aware cost model, were calculated by processing the I/O traces that were obtained by executing SysBench [15], TPC-A [16], TPC-C [16] benchmarks on MySQL and synthetic workload in the FTL simulator. The number of read, write, and erase operations for the virtual flash memory were subsequently measured using the FTL simulator. The FTL simulation results were obtained based on the flash memory operation time listed in Table 1. Table 4 summarizes the parameters of each benchmark program. The results of the flash-aware cost model were compared with the results of the FTL simulation to verify the cost model. The I/O trace extracted from the FTL simulator was processed in the flash-embedded board, while the required time was measured using the DAQ device. The required time that was measured in the flash-embedded board was compared with the calculated results from the FTL simulation to verify the simulation results.

Table 4. Parameters of the benchmark programs.

Benchmark	Parameters
SysBench	database size 120 MB
TPC-A	account 100,000, teller 10, branch 1, history 2,592,000
TPC-C	warehouse's cardinality 20, district 10, customers 3,000 items 100,000, orders 3,000, stock 100,000, new_orders 900 20 terminals per warehouse, warehouse's selectivity 20
Synthetic	80% clustered data, 20% non-clustered data 20% random & 80% sequential read/write I/Os

Table 5. Parameters of the experiments.

Parameters	Data
Size of relation *r* (KB)	500
Size of relation *s* (KB)	50, 100, 200, 400, 600, 1000, 5000, 10000, 20000
Size of buffer (KB)	100
Size of a record (byte)	100
Join algorithms type	BNLJ, MJ, HJ, INLJ
FTL algorithms	FMAX, MITS, LOG, Superblock, FAST

Table 5 lists the parameters of the experiment. The size of relation *r* was fixed while the size of the relation *s* was increased from 50KB (smaller than the buffer size) to 20 MB. Three log blocks are used in LOG, and one sequential log block and two sequential log blocks are used in FAST. The cost and time spent according to the size of relation *s* were measured.

6.1 Estimated Results Based on Flash-Aware Cost Model

In this section, the estimated results of the minimum cost join algorithm according to the size of the relation are presented based on the aforementioned flash-aware cost model. As show in Fig. 3, SysBench, TPC-A, TPC-C benchmarks were processed on MySQL and synthetic workload was processed on the simulator directly to determine α and β for the flash-aware cost model. The read and write I/Os transferred to the storage device when SysBench, TPC-A and TPC-C were operated on MySQL were extracted with trace, and processed in the FTL simulator. SysBench benchmark was run by creating an environment that occurred when multiple users perform queries on a client-server database. TPC-A and TPC-C benchmarks are generally used for evaluation of commercial database performance. In addition, a synthetic workload was created. The synthetic workload was composed of 20% sequential I/Os and 80% random I/Os. Table 6 shows the measured values of α and β for each FTL algorithm.

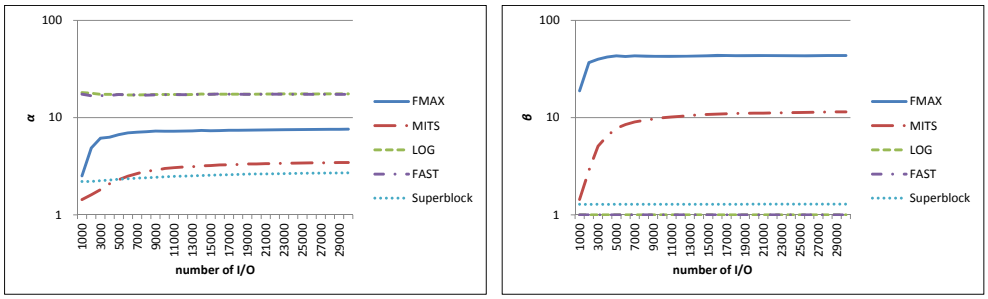
The α and β values are affected by the I/O patterns and depend on how long I/O has been performed on the flash memory. However, if I/O operations are performed for somewhat long time, the α and β converge to a specific value. Fig. 4 shows the change of α and β values according to the number of I/O operations extracted by each workload. As shown in the Fig. 4, if I/O operations are performed for more than a given number of times, α and β converge to a specific value. Table 6 shows the values measured after repeating the trace multiple times on flash memory; how α and β converge.

Table 6. α and β values in flash memory measured using Synthetic, SysBench, TPC-A benchmark and TPC-C benchmark (large block, database page size: 2KB).

FTL	Syntheic		SysBench		TPC-A		TPC-C	
	α	β	α	β	α	β	α	β
FMAX	7.72	43.45	13.90	33.77	13.52	51.80	8.89	43.80
MITS	3.57	11.84	3.21	5.31	4.56	12.03	4.26	8.62
Superblock	2.82	1.29	1.73	1.06	1.25	1.21	2.12	1.26
FAST	17.46	1.00	10.13	1.00	5.60	1.00	15.30	1.00
LOG	17.48	1.00	10.43	1.00	3.48	1.00	12.89	1.00

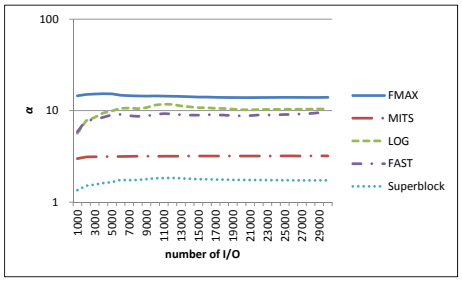
According to α and β values, FTL algorithms can be divided into three types. FMAX and MITS reserve one half of the entire flash memory space as extra space to avoid overwriting pages in the flash memory. LOG and FAST keep some blocks called as log blocks to avoid overwriting operations. Superblock improves space utilization by placing adjacent blocks in a superblock while page-level mapping is used in superblocks. The α of LOG and FAST is larger than other FTL algorithms because the hit ratio of log blocks decreases when the ratio of small random writes is being increased such as database systems. The utilization of the LOG and FAST is very low, and continuous merge operations are occurred because of many small random write operations. This result is due to the small number of log blocks. The increase in the number of log blocks in FAST contributed to FAST's better performance. Especially, with the increase in the number of random log blocks in FAST, FAST performs better than LOG [7]. FMAX and MITS write overwritten pages on the extra space of a block during random writes, α is relatively smaller than β. This is because many pages in a block have to be scanned to find the desired page since a page is written using out-of-place method on extra space. We selected the α and β values measured using synthetic workload for the cost estimation and simulation.

Fig. 5 shows the estimated costs of each join in the disk-aware and flash-aware cost models. The horizontal axis represents the size of relation *s*. The size of relation *r* was fixed at 500KB. The costs in the disk-aware cost model and flash-aware cost model are the total operation time required. We used the average seek time as 13ms in the disk-aware cost model, and the time specified on Table 1 was applied on flash-aware cost model.

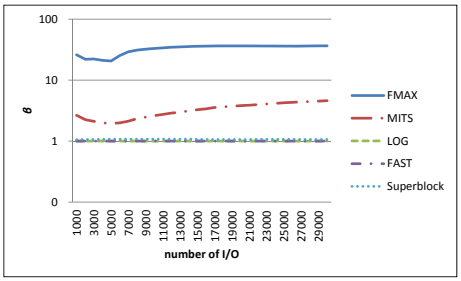


(a) Synthetic workload (α) (b) Synthetic workload (β)

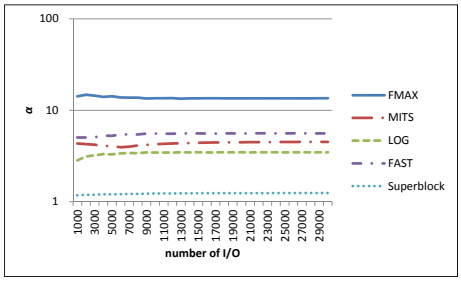
Fig. 4. Transition of α and β with respect to increasing the number of I/O operations (large block, database page size: 2KB).



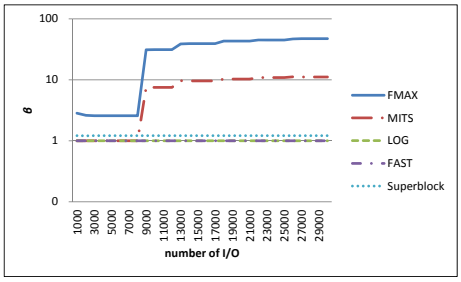
(c) SysBench benchmark (α)



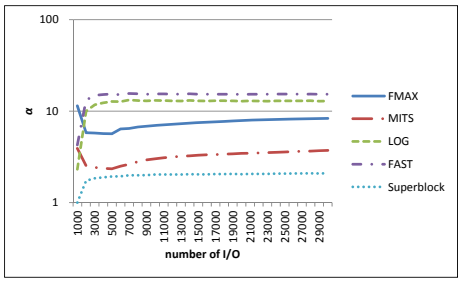
(d) SysBench benchmark (β)



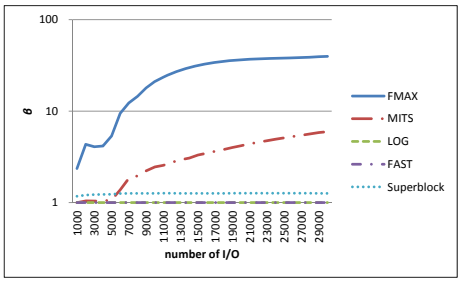
(e) TPC-A benchmark (α)



(f) TPC-A benchmark (β)

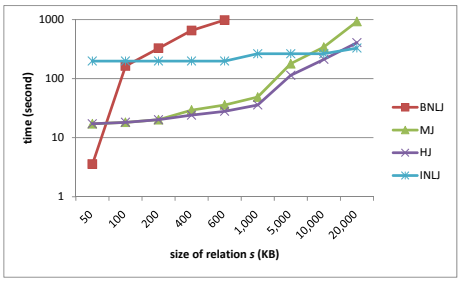


(g) TPC-C benchmark (α)

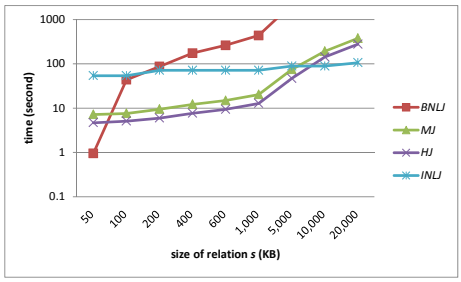


(h) TPC-C benchmark (β)

Fig. 4. (Cont'd) Transition of α and β with respect to increasing the number of I/O operations (large block, database page size: 2KB).

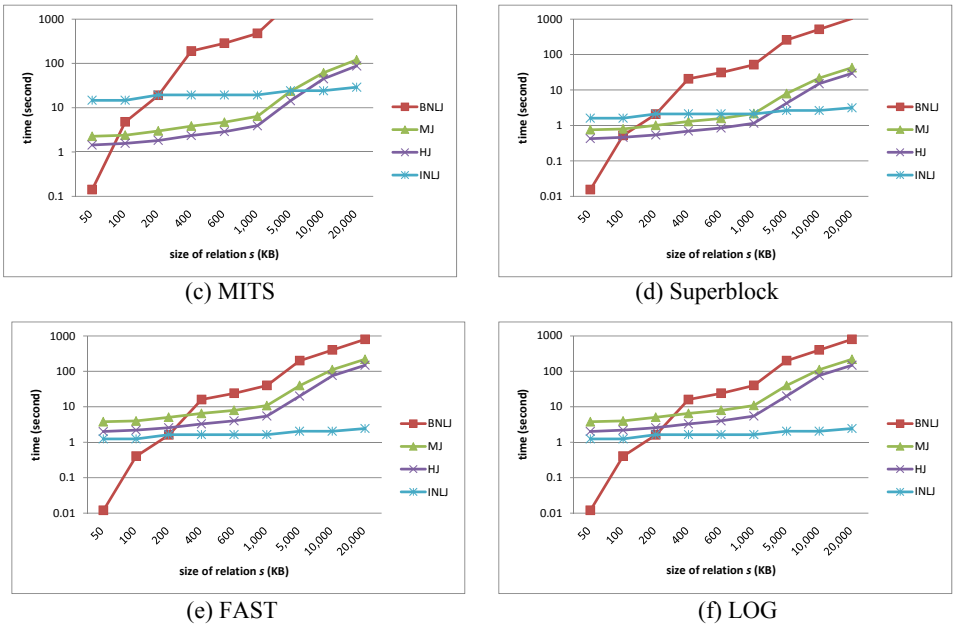


(a) DISK



(b) FMAX

Fig. 5. Estimated results for disk-aware and flash-aware cost models (large block, database page size: 2KB).



(c) MITS (d) Superblock

(e) FAST (f) LOG

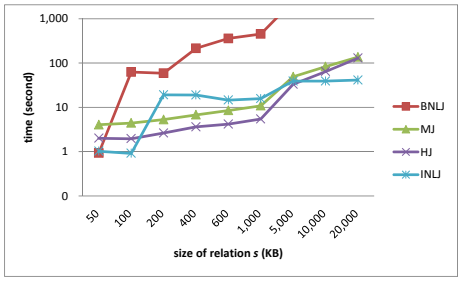
Fig. 5. (Cont'd) Estimated results for disk-aware and flash-aware cost models (large block, database page size: 2KB).

When the size of relation s is 20,000KB, both HJ and INLJ have low costs in the disk-aware cost model. In the flash-aware cost model, the cost curve of INLJ is very different from those for other join methods. If the flash memory is used as a storage device, the join operation that performs queries in the shortest time must be selected according to the results shown in Fig. 5.

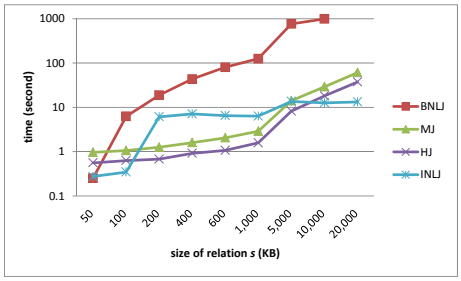
In both the disk-aware and flash-aware cost models, when relation s is smaller than the buffer size (M), BNLJ is the lowest cost incurred because the inner relation s (smaller relation will be inner relation) can be loaded into main memory. The read operation is performed rapidly in the flash memory compared to the write operations, the join operation that produces more read operations performs faster than in the disk. Because α is larger than β in the LOG and FAST algorithms, the read operations are executed rapidly compared to the write operations. Therefore, BNLJ and INLJ, which mainly perform read operations, have less cost in LOG and FAST algorithm than in FMAX and MITS algorithms. HJ and MJ, which produce write operations, have higher costs than INLJ in LOG and FAST. In the FMAX and MITS algorithms, wherein β is larger than α, BNLJ and INLJ require longer times than in other FTLs. The flash-aware cost model for the join operation is influenced more by β, which is the ratio of the additional read cost, than by α, which is the ratio of the additional write cost. Therefore, to improve the performance of FTL in terms of the join operation, the required time must be shortened by reducing the additional read cost β and the additional write cost α.

In case that the size of relation s is about 50KB, BNLJ has the least cost in both the disk-aware and flash-aware cost models. INLJ is advantageous when the relation with the index is very large and a very small relation is joined to it. INLJ performs well in an FTL

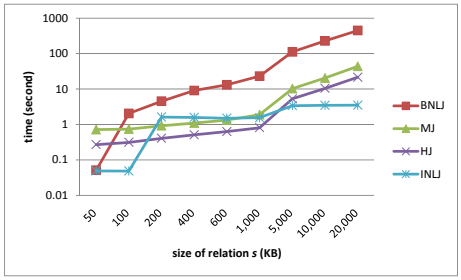
which can perform the read operation quickly. BNLJ has the lowest cost in the best case, but its cost abruptly increases with increasing *s* values.



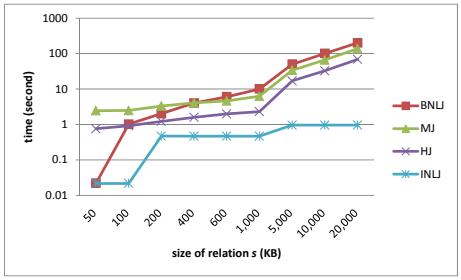
(a) FMAX



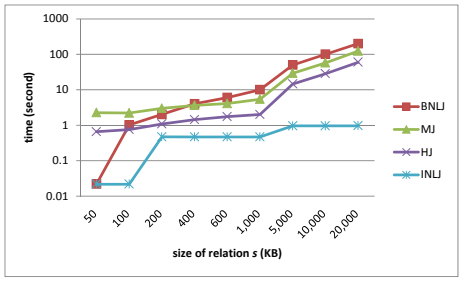
(b) MITS



(c) Superblock

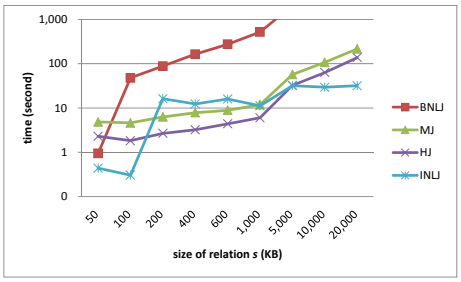


(d) FAST

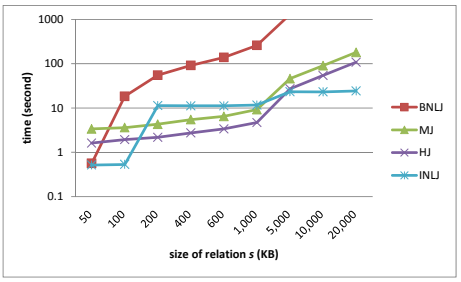


(e) LOG

Fig. 6. FTL simulation results (large block, database page size: 2KB).

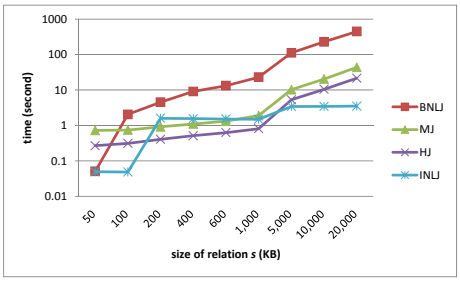


(a) FMAX

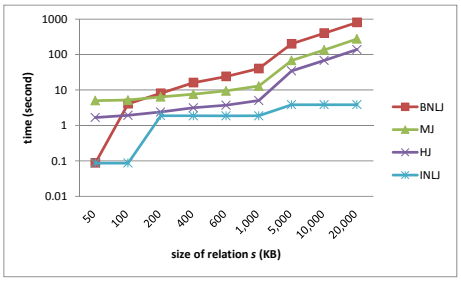


(b) MITS

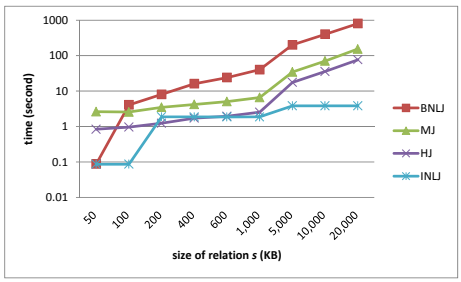
Fig. 7. FTL simulation results (small block, database page size: 2KB).



(c) Superblock



(d) FAST



(e) LOG

Fig. 7. (Cont'd) FTL simulation results (small block, database page size: 2KB).

In order to compare the predicted results using the cost model with the required time in the virtual flash memory in the FTL simulator, the following test was performed.

6.2 FTL Simulation Results

In order to obtain the FTL simulation results, the I/O trace produced from the query processing simulator was extracted. The extracted trace was run in the FTL simulator to obtain the numbers of read, write, and erase operations. The results in this section represent the total required time that was calculated in the virtual flash memory by multiplying the numbers of read, write, and erase operations and the costs listed in Table 1.

If it is assumed that all flash memory blocks are clean, the result can be distorted with decreased costs since no overwrite operations occur. Thus, to get correct answers, lots of read and write operations are performed on flash memory to converge α and β using synthetic workload as shown in Fig. 4, which is called aging.

The FMAX, MITS, LOG and FAST were originally designed for small block flash memory, we extended these algorithms for large block flash memory. Figs. 6 and 7 show the experimental results on the large block flash memory and the small block flash memory, respectively. The performance on the large block flash memory is slightly better than that of small block flash memory. Because the page size of the flash memory and the block size of database are identical in this case, some additional read and write operations could be reduced. It is the reason why the performance of the large block flash memory is better.

To obtain the experimental results of Figs. 6 and 7, the block size of database was fixed to 2KB. Generally, the block size of embedded database is small such as 1KB or

2KB. In this study, experiments were performed at both block sizes of 1KB and 2KB; there was little difference in the results while the 2KB size performed slightly better. The sectors are more clustered in case of 2KB block size, so this reduces the number of additional read and write operations slightly.

When comparing to Figs. 5 and 6, the results are similar except that the size of relation *s* is smaller than 100KB. In INLJ, it is assumed that a B-Tree is created for relation *s*. Here, the B-Tree has to be traversed to find joinable records; a visit on a node of the tree occurs a read operation. However, if there is sufficient buffer, the root node and some internal nodes of the tree can be existed in the buffer with high hit ratio. If the relation *s* is smaller than the buffer size, the tree becomes very small and the performance could be very good.

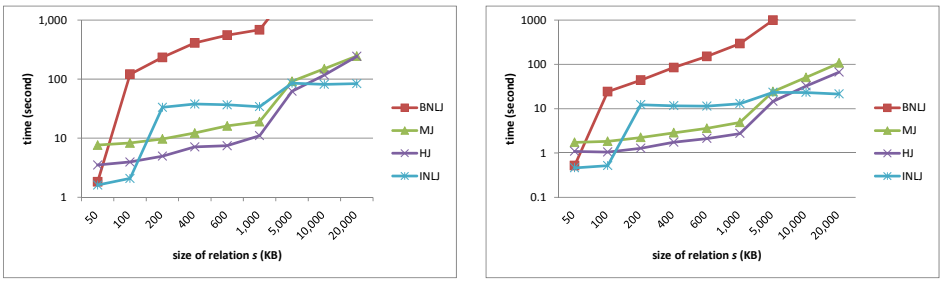
6.3 Results on Flash Embedded Board

Figs. 6 and 7 show the results of the application of the flash-aware cost model based on α and β in Table 6. In all the five FTLs, the results obtained from the FTL simulator (Figs. 6 & 7) and the cost model estimation (Fig. 5) were similar. This can be considered as a verification of the cost model's suitability. Query will be processed faster if the minimum-cost query plan is selected using the flash-aware cost model.

In order to verify the results of the FTL simulator, the I/O traces of each join operation were run in the flash-embedded board (Fig. 1), and each time was measured using the DAQ device (Fig. 2). The trace executor of the flash-embedded board performed the I/O trace in the flash memory, and the DAQ device measured the time of the read, write, and erase operations as produced from the flash memory according to the I/O trace. It was assumed for the costs of INLJ that the B+ tree index was constructed for relation *s*.

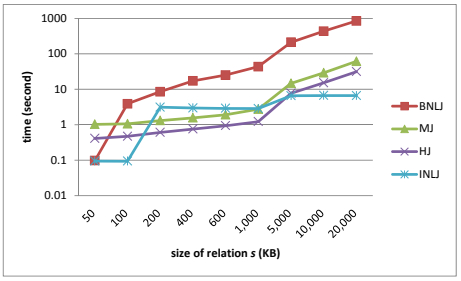
Fig. 8 shows the I/O trace of the query performance simulation as measured in the embedded board; these results were similar to the results obtained from the FTL simulator because of the same reason mentioned in section 6.2.

In the flash-embedded board, the required time was 153μs per 2KB for the read operation, 275μs per 2KB for the write operation, and 1,814μs per 128KB for the erase operation. Compared with the operation time listed in Table 1 (which was used in section 6.2), the read operation speed was much lesser. This is because the flash-embedded board stores the flash memory operation results in bulk RAM, subsequently, the re-

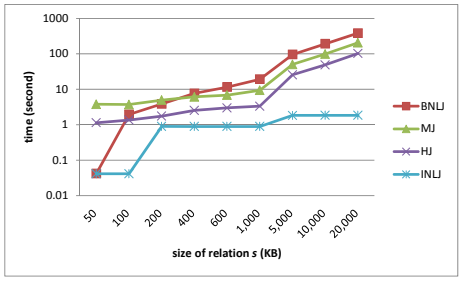


(a) FMAX (b) MITS

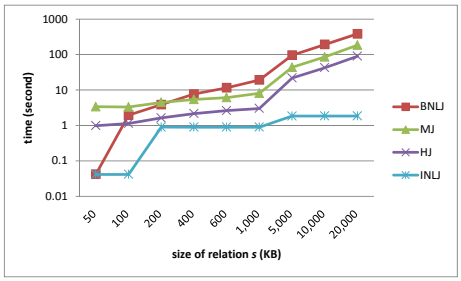
Fig. 8. Required time measurement results in the flash-embedded board (large block, database page size: 2KB).



(c) Superblock



(d) FAST



(e) LOG

Fig. 8. (Cont'd) Required time measurement results in the flash-embedded board (large block, database page size: 2KB).

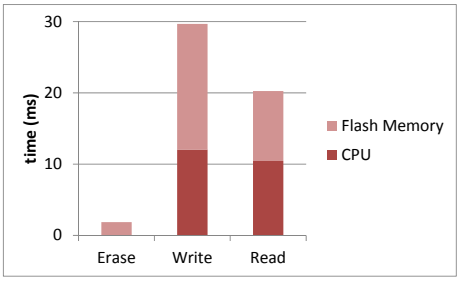


Fig. 9. Read, write, and erase time for one block from the flash-embedded board.

sults are transferred to the operating system. Fig. 9 shows the measured time of the read, write and erase operations for the data that correspond to one unit to be erased in the flash memory in the flash-embedded board. The graph shows the time spent in the flash memory and the CPU time required for the flash memory results to be transferred to the operating system. The read time was longer than that in Table 1, and the read operation was slower. The write operation takes more CPU time than the read operation because the CPU is occupied for a longer time during writing.

The overall join operation results in the estimated cost model results, FTL simulator results, and flash-embedded board results were similar. Based on the flash-aware cost model used in this paper, the query can be optimized by considering the characteristics of the flash memory and the FTL.

7. CONCLUSION

Flash memory has attracted considerable interest as a storage device for many purposes. FTL is essential to the efficient use of flash memories. Databases are used in diverse areas; embedded databases with flash memory as their storage device are expected to become popular in the near future. The characteristics of the flash memory differ from those of conventional hard disks; therefore, the query optimization method for the flash-aware database must be modified to accommodate these characteristics.

The existing cost model is defined based on the disk-aware method, and it must be modified for flash memory. In the disk-aware cost model, the number of I/Os that call for read and write operations is interpreted as the cost. In the flash-aware embedded database, the read and write requests are replaced by the read, write, and erase operations for the flash memory according to the FTL. The cost must be calculated by considering the overhead α and β for the write and read operations, respectively. In this study, we examined the cost model for the join operation of a database.

If the parameters α and β for the flash memory are unknown, the database must gradually calculate α and β to estimate the suitable cost of the flash memory storage device. In this case, after setting the initial values from experience as α and β, the time of the read or write operation to a page must be measured to gradually determine α and β with precision.

In this study, we examined the cost model only for the join query. The model needs to be analyzed for other operations in the future. In addition, it is required to minimize the power consumption for the embedded database; therefore, it is necessary to develop a cost model that minimizes power consumption.

REFERENCES

1. Samsung Electronics, Nand Flash Memory & Smartmedia Data Book, 2007.
2. A. Ban, "Flash file system," United States Patent, No. 5,4040,485, Apr. 1993.
3. B. Kim and G. Lee, "Method of driving remapping in flash memory and flash memory architecture suitable therefor," United States Patent, No. 6,381,176, 2002.
4. A. Ban, "Flash file system optimized for page-mode flash technologies," United States Patent, No. 5,937,425, 1999.
5. T. Shinohara, "Flash memory card with block memory address arrangement," United States Patent, No. 5,905,993, 1999.
6. J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A space-efficient flash translation layer for compact flash systems," *IEEE Transactions on Consumer Electronics*, Vol. 48, 2002, pp. 366-375.
7. S. W. Lee, D. J. Park, T. S. Chung, W. K. Choi, D. H. Lee, S. W. Park, and H. J. Song, "A log buffer based flash translation layer using fully associative sector translation," *ACM Transactions on Embedded Computing Systems*, Vol. 6, 2007.
8. J. U. Kang, H. Jo, J. S. Kim, and J. Lee, "A superblock-based flash translation layer for NAND flash memory," in *Proceedings of the 6th ACM & IEEE International Conference on Embedded Software*, 2006, pp. 161-170.
9. S. E. Wells, "Method for wear leveling in a flash EEPROM memory," United States

Patent, No. 5,341,339, 1994.
10. E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Computing Surveys*, Vol. 37, 2005, pp. 138-163.
11. T. Chung, D. Park, S. Park, D. Lee, S. Lee, and H. Song, "A survey of flash translation layer," *Journal of Systems Architecture*, Vol. 55, 2009, pp. 332-343.
12. P. Estakhri, and B. Iman, "Moving sequential sectors within a block of information in a flash memory mass storage architecture," United States Patent, No. 5,930,815, 1999.
13. S. J. Kwon and T. Chung, "An efficient and advanced space-management technique for flash memory using reallocation blocks," *IEEE Transactions on Consumer Electronics*, Vol. 54, 2008, pp. 631-638.
14. P. A. Silberschatz, H. K. Korth, and S. Sudarshan, *Database System Concepts*, 4th ed., McGraw-Hill, 2002, pp. 503-514.
15. SysBench: a system performance benchmark, http://sysbench.sourceforge.net.
16. J. Gray, *The Database Handbook: for Database and Transaction Processing Systems*, Morgan Kaufmann, CA, 1993.

Sangwon Park received the B.S. and M.S. degrees in the Computer Engineering Department from Seoul National University, February 1995 and February 1997, respectively, and the Ph.D. degree in School of CS&E from Seoul National University, February 2002. He is currently an Associate Professor in Hankuk University of Foreign Studies. His research interests include flash memory based DBMSs, multimedia databases, and mobile phones.