IADIS International Journal on Computer Science and Information Systems
Vol. 3, No. 1, pp. 59-70
ISSN: 1646-3692

# PERFORMANCE IMPROVEMENTS OF MR-TREE OPERATIONS ON NAND FLASH MEMORY

**Hyun Seung Lee, Ha Yoon Song, and Kyung-Chang Kim** ***Department of Computer Engineering, Hongik University, Republic of Korea***

**ABSTRACT**

Flash memory can be a viable solution for the future embedded systems. Embedded systems usually carry database as a part of embedded software. It is well known that Flash memory is far faster than usual hard disk storages especially for read however it takes much time for Flash memory to delete than to read or to write. For a database system, index trees on Flash memory have been widely studied for the better performance. From our previous researches on indexing tree performance on Flash memory, we suggest a delaying technique for index tree operations on Flash memory. Among B-tree, R-Tree and MR-tree, we choose MR-tree as it is multimedia database index. Considering both of the flash memory operation characteristics and MR-tree operation characteristics, we adjust MR-tree parameters for a just fit for Flash memory and applied aggregated-delayed MR-tree operations on Flash memory. We believe we find a reasonable parameter and efficient operation delaying technique for MR-tree on flash memory. The benefit of our researches has been proven by the experimental results based on several database access profiles.

**KEYWORDS**

Flash Memory, Index Tree, MR-Tree, Tree Block Size, Delayed Operation

## 1. INTRODUCTION

Nowadays ubiquitous computing environment with mobile devices, consumer electronics and so on widely adopts Flash memory system with rapid development of memory technologies. Mobile phones, PDA, smart card, digital cameras are rapidly consumed by peoples and almost of them are using Flash memory with nice characteristics such as low power, non-volatility, high performance, physical stability, portability, etc. Flash memory, as well as solid state

"This work was supported by 2007 Hongik University Research Fund"

drives (SSD), has been discussed as a future replacement of hard disk storages. And the memory characteristics of Flash memory are still under road to be developed. Usual Flash Memory (FM) consisted in several blocks with 32 sectors, one sector of 512 byte data area and 16 byte spare area. Different from Hard Disk Drives (HDD), FM has read access time comparable to Dynamic RAM (DRAM) however it has very slow write access and has no ability of overwrite. Overwriting can be done by writing after block delete. In order to solve these problems, there are FTL (Flash Translation Layer) between file systems and FM. Delete operations can be translated into mappings from logical sector address to physical memory address which has been already deleted. Such sort of FTL operations can help the use of FM with usual file system. However, most of FTL are also under further development with huge number of options.

In case of constructing database system with FM, index structures will be stored on FM while actual records will be stored on HDD. With rapid increase of FM capacity, actual records will be also ported on FM in the near future. The use of FM as index storage will naturally increase DB access performance, but it totally depends on the type of index trees and the DB access profile as well. We can discuss three types of index trees. B-tree is a data structure with efficient DB indexing. R-tree is for spatial data access index with multimedia DB. Also, we can consider MR-tree as an upgraded version of R-tree. These index trees must be updated with Insert, Delete, or internally updated by Split operations. Once we implement HDD based B-tree on FM, a direct porting may cause performance degradation. Most of past researches concentrated on delete operations and improved with buffering such as BFTL [11] or BOF [7].

But both of improvements lead in overhead of read operation and management overhead of buffers with data.

In addition, the limitation of the number of write operations on FM will require index update to be evenly distributed over blocks in order to fully utilize FM life cycle. Under these situations, we will find the MR-tree parameters for FM and will consider aggregation of the effect of MR-tree operations for FM characteristics. We believe this approach will inform researchers a key to improve their database systems on FM.

Following sections are consisted as follows. In section 2 we will discuss FM, index trees and related works in detail. The next section contains our key ideas regarding MR-tree operations on FM in terms of performance. Then we will show several experimental results in section 4. In the final section, we will conclude our work and discuss about the future possibilities of MR-tree implementation on FM.

# 2. RELATED RESEARCH

## 2.1 Flash memory characteristics

We focus on the NAND FM since the NAND flash is one of the prominent FM currently under commercial manufacturing. The access characteristics of NAND FM are as follows [3, 13]:

- 1.2 microsecond for 2 Byte read
- 35.9 microsecond for 512 Byte read
- 201 microsecond for 2 Byte write

- 226 microsecond for 512 Byte write
- 2 millisecond for 16 KB block delete

Table 1 also contains comparisons between various memory devices including NAND FM. NAND FM has operations of read and write on the unit of sector, and write operation must be done after deleting the corresponding block. Delete operations are done in the unit of block, thus it takes far more time than read and write operations which can be done in the unit of sector. Because of this difference between the unit of write and delete operations, write operations cannot be done immediately. Write-pending pages should be copied on other blank pages with modifications and this specific operation is called not-in-place update. Free page is a clean page never has been written. Dead page is a page having out-of-date data after not-in - place update. Live page is a page having up-to-date data after not-in-place update [2]. The number of dead pages will increase with cumulated write operations and the number of free pages will be degenerated. Therefore garbage collection will be required.

Table 1. Memory device access characteristics

| Media | Read | Write | Erase |
|---|---|---|---|
| **DRAM** | 60ns(2B)<br>2.56us(512B) | 60ns(2B)<br>2.56us(512B) | - |
| **NOR Flash** | 150us(2B)<br>14.4us(512B) | 211us(2B)<br>3.53us(512B) | 1.2s<br>(128KB) |
| **NAND Flash** | 10.2us(2B)<br>35.9us(512B) | 201us(2B)<br>226us(512B) | 2ms<br>(16KB) |
| **Disk** | 12.4ms(512B)<br>(average) | 12.4ms(512B)<br>(average) | - |

We need block recycling policy to choose a specific dead page deleted and converted to free page. Also we need wear leveling in order for uniform selection of dead page to be deleted since there are limitations in the number of block delete. Thus it is clear that write operation requires extra cost with wear leveling and garbage collection. Flash Translation Layer (FTL) helps file system to recognize FM as a logical space such as that of HDD. FTL mapping algorithms are classified as sector mapping, block mapping and hybrid mapping. Also, the logical-physical address translation is classified as in place and out of place where physical sector first for this while logical sector first for that. For example, FAST(Full Associate Sector Translation) [5], which improved log block mechanism, logically partites FM memory space into data block, log block, and sequential write block. It marks "in-place" on a data block for a sector written with hybrid mapping. Once an overwritten takes place for this sector, it choose a specific sector on log block and marks it as "out-of-place," except the zero-th sector be overwritten. Instead, a sequential write block is chosen and then marked as "in-place". Once a zero-th sector be overwritten again, a merge operation happens or an exchange operation happens in order to minimize delete operation. There exist several log blocks where there could be FIFO merge operation if there are no free sector on a log block. FAST supports mapping between file system and FM efficiently and file system can recognize FM as block device such as hard disk drive.

## 2.2 Index Trees

B-tree [8] is an index most representatively used in database management system that is based on disk and main memory. Operations of insert, delete, search of data are done easily and B-tree supplies efficient balance algorithm.

R-tree [1] is guided hierarchic data structure from B-tree structure for spatial data access such as multimedia data. Each node is composed of smallest d-dimension quadrilateral including its own child node. Leaf node includes pointer for actually geometrical object instead of including child nodes in database. Objects are expressed by minimum sorted quadrilateral in which they are included. Because R-tree is height balanced tree, it announces node split to parent node, so approximately average 70% of entries have filled in leaf node. R-tree often does to increase height of tree for height balance of leaf nodes, and it causes unnecessary cache miss finally. MR-tree [4] uses basic index structure of R-tree, but height among sub trees can be unbalanced and the difference is restricted at most one. If height between sub trees is restricted more than one, it runs height balance algorithm. And, it has special feature that splits are delivered to parent node when split is produced. In case of a node exists which has empty entry on insert operation and it judges re-insertion possibility electively in occasion of delete. MR-tree structure is shown in figure 2.

Index nodes of MR-tree are classified to internal node, leaf node and half-leaf node. Half-leaf node means a node that have all entries for leaf nodes and data object. If new object is inserted in the half-leaf node continually, this node is changed to an internal node that has height of two. Node structure of MR-tree other than R-tree shows two distinguished additional field. Height field has a one-increased value of the biggest heights of child nodes to field that display height of node, and is used to search the imbalance state of MR-tree height.

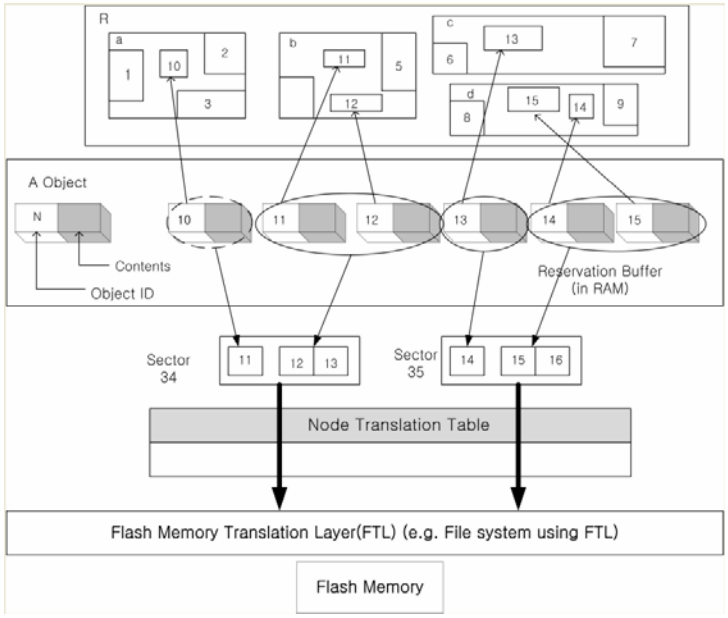


Figure 1. R-tree on FM

Data entry number field is used by half-leaf node and stores number of data entry stored in the node. Split of leaf node is delivered to parent node only if there is empty entry in internal node on insert operation. Therefore, it can certainly reduce whole number of node splits. MR-tree shortens tree height and indicates good performance in search by decreasing the size of MBR of internal nodes. It improves entry space utilization for internal node by delivering a node split to parent nodes by use of more than one empty entry among internal nodes on insert operation, controls increase of tree height maximum, and reduces the count of cache miss.

## 2.3 Flash Memory Index

The major reason of FM performance decaying in the indexing of huge multimedia database is a set of random accesses rather than a set of sequential accesses in case of index modifications. For example of tree index, even a modification of one tree node will cause the modification of data of that entry, address and key values pointing to data, and pointer values to parent and child nodes. Those modifications of index lead in modifications of related FM pages. It eventually increases dead pages and decreases free pages, cause large number of delete operations and degenerates the life of FM. Thus we must consider FM characteristics in order to operate index trees efficiently on the FM.

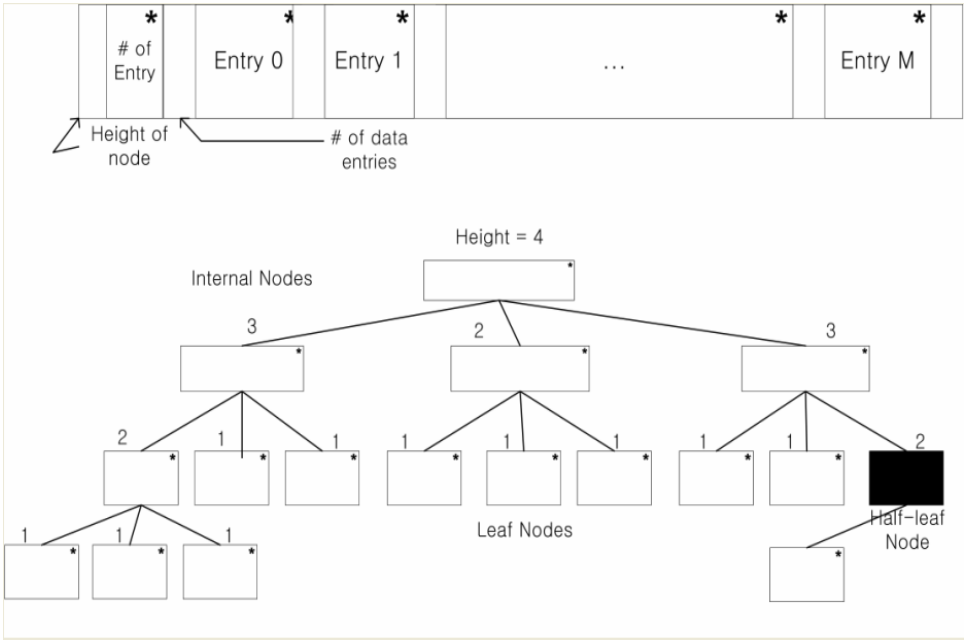


Figure 2. Structure of MR-tree

There are related researches regarding FM usage for index trees. B-tree is for BFTL[11] and R-tree is for [12], respectively for FM index trees. These two similar approaches contain a scheme of reservation buffer and index unit. Every modification of index trees creates index

unit and reserved in reservation buffer. A fully contained reservation buffer ultimately transferred to FM pages.

Node translation table manages information of sectors where node data are contained, since one node can be distributed over several sectors of FM. Compressions are executed in case node translation table became large. Here, the frequent compression can be overhead as well as intermediate search of node translation table in case of node search. BOF[7] overcomes this situation by storing index of one node per one sector. Without the use of node translation table, it eliminates compression overhead and enhances search performance.

Even though the presence of previous research, researches on spatial index for multimedia or biomedical databases which usually contains huge data are still required in order to fully utilize the characteristics of FM, a.k.a., fast access time, portability, lifetime limitation and so on. Especially, researches are required for index structures, node sizes, and the combination of those two with the characteristics of FM access unit for the efficient operation of index trees.

Thus, with the introduction of MR-tree structures and operations combined with FM characteristics will be an idea for the performance purpose. We introduced “delaying of MR-tree index operations” and “consideration of node size for MR-tree nodes.” Next section shows the details on this issue.

# 3. MR-TREE ON FLASH MEMORY

## 3.1 Considerations on Node Size

The structure of MR-tree reduces MBR (Minimum Bounding Rectangle) of intermediate nodes as well as tree height with the full usage of intermediate nodes rather than leaf nodes.

It is notable that with different node sizes of MR-tree for the same number of index, the structures of MR-trees are totally different. Thus MR-trees with different structures shows different performance as the timings of search, insertion, and deletion operations are varied.

With the 16byte size of MBR for R-tree and MR-tree, the search time rapidly decreases as the size of nodes increase while it increases when it passes the minimum [4]. With node size between 256byte and 512byte, the search performance shows optimum results. It is because the balance between tree height and node size. The smaller node size results in high tree and large search depth, and the bigger node size also results in high cache miss without reducing the number of nodes to access for search.

Table 2 shows the tree characteristics for varying typical node size. The larger node results in smaller number of nodes and small tree height, larger size of total tree, and smaller number of tree split.

Table 2. Tree characteristics with different node size over 1,000,000 insertions

| **Node Size in byte** | **128** | **256** | **512** | **1024** |
|---|---|---|---|---|
| Number of Nodes | 21153 | 3615 | 664 | 52 |
| Total size of Tree in Mbyte | 212.5 | 399.8 | 812.9 | 1532.7 |
| Tree height | 11 | 8 | 5 | 4 |
| Number of tree Split | 241020 | 118494 | 57195 | 28092 |

We used one million number of spatial data insertion. Rectangle data information is created with the insertion of MR-tree, and one tree node contains from 6 rectangles to 51 rectangles. The larger node results in high search performance due to the larger number of data per one node and reduces the number of split and tree height. However the larger nodes results in lower spatial utilization and thus shows the bigger size of total tree. Of course, tree height and width affects the search performance. We must consider optimal height of tree.

From the considerations of search performance and node size above, and considerations of FM access characteristics, we can choose the node size of 512byte in order to balance between tree characteristics and FM access unit. We expect that MR-tree shows better search performance as well as insertion and deletion with the node size of 512byte. FM also adopts 512byte as a default unit of read and write.

## 3.2 Delaying MR-tree operations on Flash Memory

We decide 512byte as a node size of MR-tree and it fits the access unit of FM. From this approach we can reduce write cost of FM for MR-tree operation which is a major cause of performance degradation. However, there are still multiple factors for write operation on FM from the view of MR-tree.

Once we apply node modifications of index, insertions and deletions of tree nodes will be operated. These relatively small operations which may not fill the whole sector of FM especially for write access and they will increase dead page and decrease free page for FM. From the considerations of MR-tree index, nodes have pointers to parents and child as well as keys and data information. Even one of the modifications in a tree node will cause a whole sector write to FM. If we have to change the pointers to parents or child, this modification will effect consequently to the adjacent nodes. Such a large number of modifications eventually degrade MR-tree performance on FM. Therefore, we suggest a delaying technique of MR-tree operations, especially modifications, on FM.

We must gather small operations till the whole size of small operations will fill the sector size of FM. This aggregation of small size operations will leads in delaying of operations of MR-tree. From this approach, we can decrease the number of dead pages, and increase the number of free pages and lifetime of FM as well. In addition, the coincidence of tree node size of FM sector size will leads in low overhead in node information management.

Figure 3 depicts a structure of MR-tree for delaying operations. Especially index modifications will be delayed if the modifications require smaller memory amount than the size of a FM sector.

In our approach, not every result of modifications will be applied to FM directly. Instead, it will be gathered together till the required memory size will be met as the size of a FM sector.

Intermediate buffers which have the exact size of FM size, called sector buffer, will be applied in the middle of modification operations. Tree index do not access FM sectors directly, however, they access sector buffers. Once a sector buffer is full with modifications, the modifications to the sector will be applied to a FM sector. This delaying will effectively help high utilization of FM.

Another benefit of this approach is that it does not requires node translation table such as BTFL approach, since one tree node can be a exact match in size of FM sector. Information for one node can be managed in a unit of FM sector. This elimination of BFTL layer will cause better performance of MR-tree access.

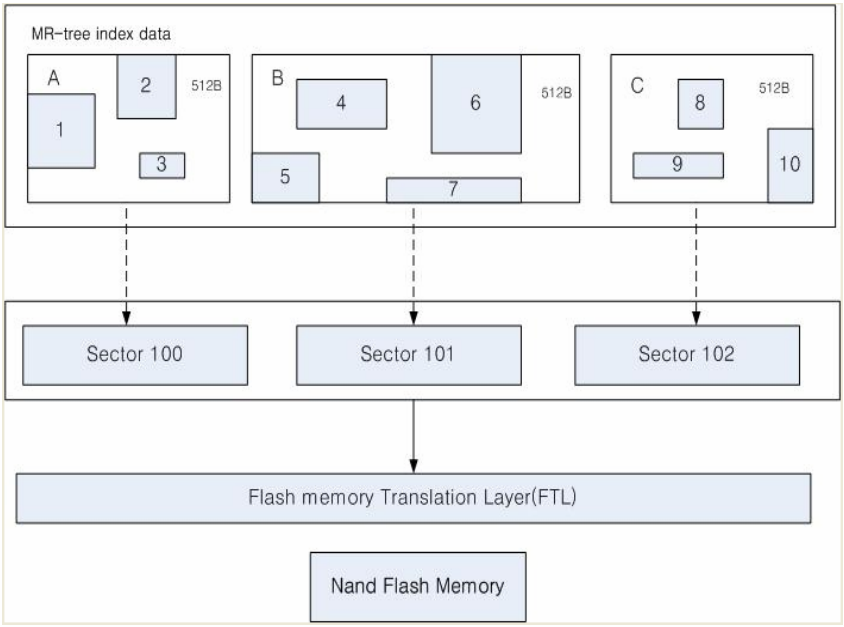


Figure 3. MR-tree structure for delaying operation on FM

# 4. PERFORMANCE RESULTS

We do various set of experiments for MR-tree operations on FM. First we design DB access profiles in order to simulate MR-tree operations. Second, we compare B-tree, R-tree, MR-tree operations by measuring performance on the same FM specifications. Measurements are also conducted for Hard disk drivers in order to prove the merits of FM for tree index over hard disk drivers. These results can be found [6]. Finally the effects of delayed operations are experimented. The effect will be shown in combination with the effect of node size.

## 4.1 Database Access Profile Design and Node Parameters

We designed database access profiles in three aspects. First, we assume usual operations of database usage, where most of operations are search access and smaller number of insertions and deletions are made. This profile is called Read Oriented Profile and composed of 8000 searches, 1000 insertions and 1000 deletions respectively. We assume another situation of initial construction stage of database where insert operations are mostly executed. This profile is called Write Oriented Profile and composed of 1000 searches, 8000 insertions and 1000 deletions respectively. Finally we assume a total shrinking situation of database. Even though it is a rare case, it is worth looking to demonstrate the effect of delete operations on tree index. This profile is called Delete Oriented Profile and composed of 1000 searches, 1000 insertions and 8000 deletions respectively.

With these DB access profiles, we can calculate and simulate the number of FM access according to the dedicated profiles. Table 3 shows the result. For each DB access profile, we

can compare the number of FM access with and without delayed operations, including the number of read, write and delete operations to FM. In case of Read oriented profile, which is a common case of database access, we reduced 46% in number of access to FM and will results it enhanced performance of MR-tree on FM. Other profiles also show considerable reduction in FM access.

## 4.2 Experimental Results

We conducted another experiment in order to prove the effect of MR-tree node size as well as the effect of aggregated and delayed operations. We implemented actual MR-tree and a set of related operations. Every experiment is executed with 60,000 operations with two dimensional rectangular data. Thus we can get performance results of MR-tree operation timings in a unit of milliseconds.

We focused on read oriented and write oriented profiles which are usual usage of database index. For every graph follows, x-axis stands for the number of operations and y-axis stands for accumulated time of operations. In order to demonstrate the effect of node size, we choose 128byte, 256byte, and 512byte for a node size respectively. And we combine the effect of node size with the effect of delayed operations. For example, 512-delay stands for 512byte node size with delayed operations. Each rectangle has a size of 40byte including data pointer and node pointer. Thus these 40byte rectangles must be gathered to fill the whole size of node, and this gathering requires an actual access to FM must be delayed for a while.

Table 3. Number of FM access for each profile

| Profile | Operation Types | R | W | D | Benefit(%) |
|---|---|---|---|---|---|
| Read Oriented | Number of Access without delay | 17252 | 3012 | 72325 | 46 |
| | Number of Access with Delay | 7956 | 878 | 30215 | |
| Write Oriented | Number of Access without delay | 3426 | 23589 | 73695 | 25 |
| | Number of Access with Delay | 854 | 6678 | 30215 | |
| Delete Oriented | Number of Access without delay | 6251 | 5697 | 422325 | 30 |
| | Number of Access with Delay | 1852 | 1678 | 172215 | |

Figure 4 shows the effect of 60,000 *insertions* to MR-tree index on FM. Without delayed operations, the effects of node size are minute while 512byte nodes show a little better performance in actual numbers. However, the effects of delayed operations are clear and then 512byte node size with delayed operations is a winner in this case.

Figure 5 shows the effect of 60,000 searches to MR-tree index on FM. Here, the effect of node size and delayed operations are distinct. With the bigger node size, the search performance increases with smaller tree height as expected. Once the effect of node size is combined with delayed operations, we can earn the best performance of database index search as well as successful management of FM.

## 5. CONCLUSIONS AND FUTURE RESEARCHES

In this paper, we proposed an efficient delaying technique of MR-tree operations on NAND Flash memory. MR-tree, as a spatial index data structure for cache awareness, is treated in order to improve its performance on NAND Flash memory. We investigated natures of MR-tree operations over various sizes of nodes. Considering Flash memory read, write and delete operation characteristics, we proposed a delay technique of MR-tree operations. The unit of read and write, a sector of Flash memory, is 512byte and as expected we experienced the best performance of MR-tree operations with 512byte node size. Insertion, deletion and search operations can be applied to Flash memory can be delayed after the aggregation of related data in a size of Flash memory sectors, and the cost of write operations can be reduced considerably.

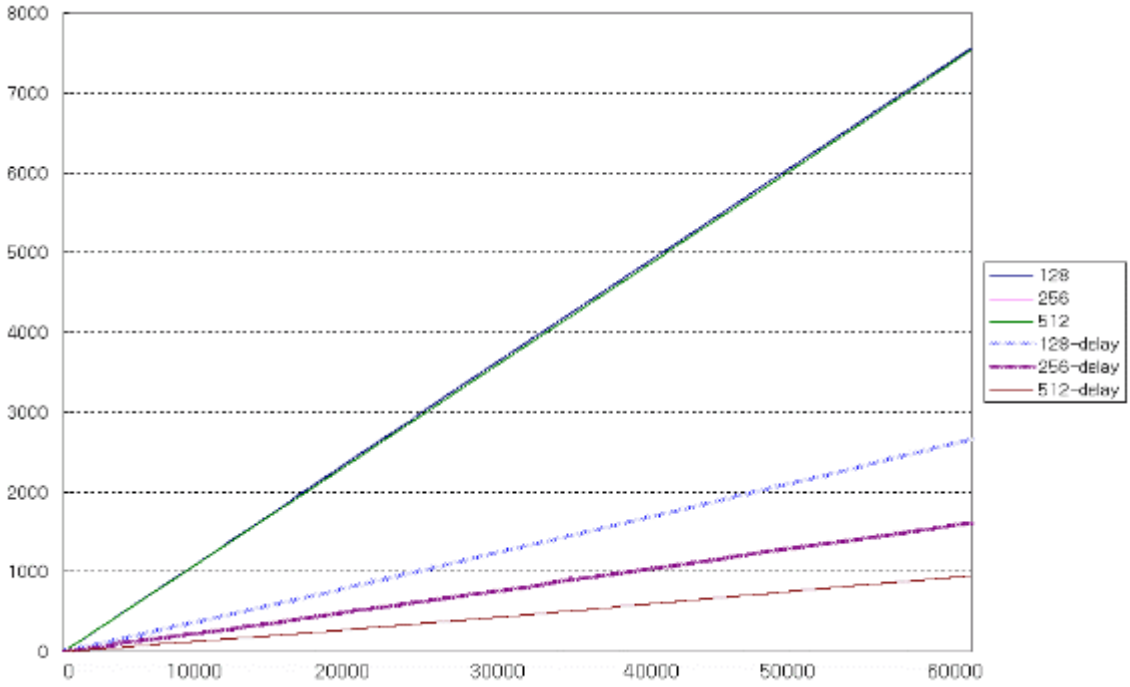


Figure 4. Effect of insertions on different size of nodes

The optimized MR-tree for flash memory with node size of 512 byte shows better performance in search operation which is related in read operations, and the write and delete operation in a unit of 512 byte shows high efficiency in performance and node management. As a result, we minimized write and delete operations to Flash memory which leads in nice performance of MR-tree operations and it extends the lifetime of Flash memory which has lifetime limitation because of delete operations. Thus we also delayed the degeneration of Flash memory lifetime.

For the following research, wear-leveling of Flash memory can be a candidate. We can set the sector size of Flash memory to the node size of MR-tree and identify the optimal page size of wear-leveling. It would improve the performance of wear-leveling for MR-tree.

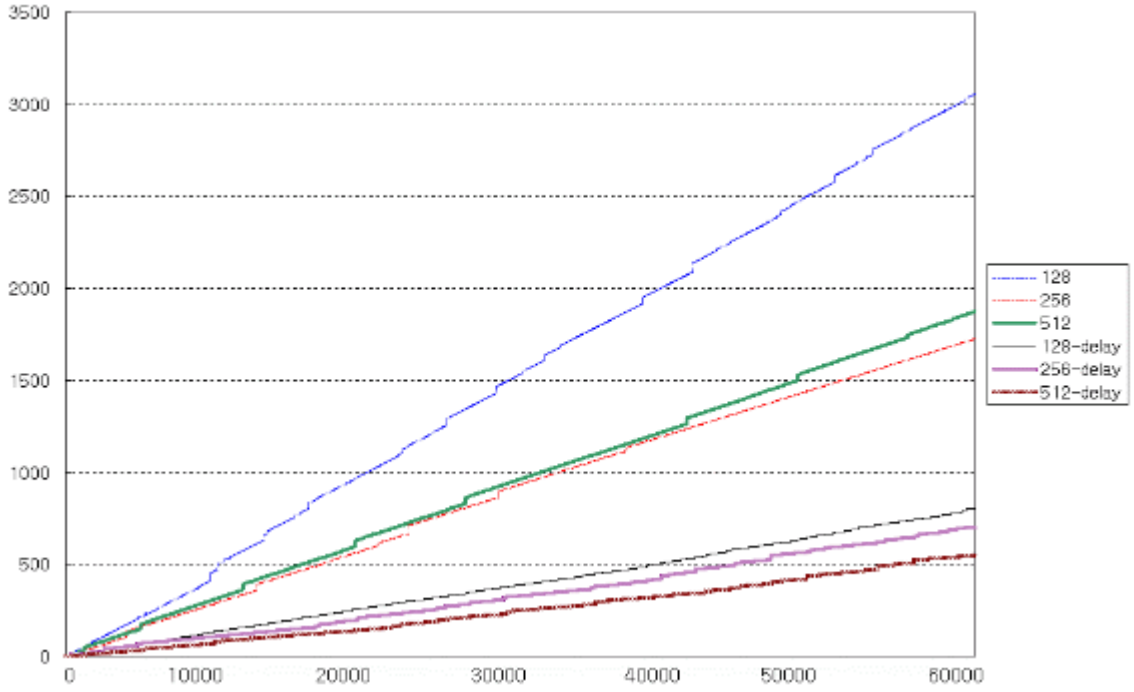


Figure 5. Effect of searches on different size of nodes

## REFERENCES

1. Antomn Guttman, 1984. R-trees - a dynamic index structure for spatial searching. In International conference on management of data proceedings of the 1984 ACM SIGMOD , pages 47–57.
2. Eran Gal . et al, 2005. Algorithms and data structures for flash memories. In ACM Computing Surveys(CSUR), pages 138–163.
3. Jesung Kim. et al, 2002. A space efficient flash translation layer for compact flash systems. In IEEE Transactions on Consumer Electronics, pages 366–375.
4. Kyung-Chang Kim. et al, 2004. MR-Tree : a cache-conscious main memory spatial index structure for mobile GIS. In Web and wireless geographic information systems, 4th international workshop (W2GIS 2004), pages 167–180.
5. Sang-won Lee. et al, 2005. A log buffer based Flash Translation Layer using fully associative sector Translation. ACM transactions on Embedded Computing systems.
6. Hyun Seung Lee. et al, 2007. Performance of Index trees on Flash memory. International Conference in Principles of Information Technology and Applications(PITA'07).
7. Jung-hyeon Nam . et al, 2005. The efficient design and implementation of the B-tree on flash memory. In Korea information science society.

8. Beng chin Ooi . et al, 2002. B-trees : bearing fruits of all kinds. In Australian Computer Science Communications, Proceedings of the 13th Australasian database conference (ADC02).
9. Chanik Park. et al, 2004. Energy aware demand paging on Nand Flash-based embedded storages. In Proceedings of IEEE/ACM international symposium on Low Power Electronics and Design(ISLPED 2004).
10. Michael Wu. et al, 1993. eNVy : a nonvolatile main memory storage system. In proceedings, Fourth workshop on workstation operating systems(WWOSIII).
11. Chin-Hsien Wu. et al, 2003. An efficient B-tree layer for flash memory storage systems. The 9th, International conference on Real-Time and Embedded Computing systems and Applications.(RTCSA).
12. Chin-Hsien Wu. et al, 2003. An efficient R-tree implementation over flash-memory storage systems. In Proceeding of the 11th ACM international symposium on Advances in geographic information systems, pages 17–24.
13. SAMSUNG NAND flash SLC-small block, 2007. In ”http://www.samsung.com/global/business/semiconductor/productInfo.do
?fmly id=158&partnum=K9F1208R0C&&ppmi=1157”.