Journal of Korea Multimedia Society Vol. 12, No. 12, DECEMBER 2009(pp. 1710-1717)

A Garbage Collection Method for Flash Memory Based on Block-level Buffer Management Policy

Liangbo Li[†], Song-Sun Shin[††], Yan Li[†††], Sung-Ha Baek[††††], Hae-Young Bae[†††††]

ABSTRACT

Flash memory has become the most important storage media in mobile devices along with its attractive features such as low power consumption, small size, light weight, and shock resistance. However, a flash memory can not be written before erased because of its erase-before-write characteristic, which lead to some garbage collection when there is not enough space to use. In this paper, we propose a novel garbage collection scheme, called block-level buffer garbage collection. When it is need to do merge operation during garbage collection, the proposed scheme does not merge the data block and corresponding log block but also search the block-level buffer to find the corresponding block which will be written to flash memory in the next future, and then decide whether merge it in advance or not. Our experimental results show that the proposed technique improves the flash performance up to 4.6% by reducing the unnecessary block erase numbers and page copy numbers.

Key words: Flash Memory, Garbage Collection, Buffer manager, Block-level Buffer

1. INTRODUCTION

Recent advances in processors, memory, storage, and connectivity have paved the way for next generation applications that are data-driven, whose data can reside anywhere and that support access from anywhere. Memory sizes have gone up and prices have come down significantly. With advances in flash memory technology, large flash drives are available at reasonable prices [1].

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 [2].

Compared to hard disk storage media with the inevitable mechanical delay in accessing data, flash memory provides fast uniform random access. However, flash memory is characterized by its erase-before-write operation; it must be erased before new data is written to a given physical location. Unfortunately, write operations are performed in unit of sector, while erase operations are executed in unit of block, usually, a block consists of many sectors [5]. Due to these features, a special software called FTL (Flash Translation Layer) is employed, which handles the algorithmic sequences of read, write and erase operations of flash

※ Corresponding Author : Hae-Young Bae, Address : (402-751) Hitech-Center 1008#, 253 Younghyun-dong Nam-gu, Incheon 402-751 South Korea, TEL : +82-32-860-8712, FAX : +82-32-862-9845, E-mail : hybae@inha.ac.kr
Receipt date : July 1, 2009, Revision date : Sep. 1, 2009
Approval date : Dec. 10, 2009
[†] Department of Computer Science & Information engineering, INHA University
(E-mail : liliangbohost@gmail.com)
[††] Department of Computer Science & Information engineering, INHA University
(E-mail : hermit@dblab.inha.ac.kr)
[†††] Department of Computer Science & Information engineering, INHA University
(E-mail : leeyeon@dblab.inha.ac.kr)
[††††] Department of Computer Science & Information engineering, INHA University
(E-mail : shbaek@dblab.inha.ac.kr)
[†††††] Department of Computer Science & Information engineering, INHA University
※ This research is supported by INHA University.

memory. The main function of FTL is to map the storage interface logical blocks to physical pages.

The buffer mechanism is also used in many PMPs (Portable Media Players). When the host system writes data to flash memory, the data will be written to buffer firstly. If the buffer has no space to accommodate new coming data, the least recently used data will be selected as victim and then flushed to flash memory. This mechanism can reduce write operations to flash memory when there are lots of hot page write commands.

In this paper, we propose a novel garbage collection method. When FTL need to merge blocks during the garbage collection steps, it will refer to the contents of buffer. By examining the contents of buffer, FTL copies the best corresponding buffer block to flash memory or selects the appropriate block as victim block to improve the performance of flash-based storage system. In shortly, we focus on three issues in this paper. First, we discuss the process that how does a block-level buffer flush data to flash memory; then, we present the novel garbage collection method; finally, we introduce the victim block selection technique.

The rest of this paper is organized as follows. In section 2, we briefly talk about the background and related work. Section 3 presents the motivation of the proposed method. Section 4 describes the details of our technique. Experimental results are presented in section 5. Finally, we conclude our work in section 6.

2. BACKGROUND AND RELATED WORK

2.1 Flash Memory

Flash is non-volatile, provides reasonable sizes at affordable prices, and significant advances are taking place in its storage. The capacities are increasing and prices are coming down [1].

Compared with magnetic disks, flash memory has many significant different characteristics.

No latency. Flash memory has no latency associated with the mechanical head movement to locate the proper position to read or write data. In magnetic disks, this seek time have been one of the most time-consuming parts in I/O activity [7].

No in-place-update: The previous data should be erased first in order to overwrite another data in the same physical area. The worse problem is that the erase operation cannot be performed on the particular data selectively, but on the whole block containing the original data [9].

Asymmetric operation costs. For flash memory, read operations are faster than write operations. Because of a write request always involves some erase operations, its entail behavior will cost longer.

Uneven wear-out. After specified number of write/erase operations, the blocks of flash memory will be worn out. Once the number is reached, the block cannot be used any more. Therefore, flash memory requires a well-designed garbage collection scheme to evenly wear out the flash memory region.

2.2 Flash Translation Layer

Different types of FTL algorithms exist. The main goal of FTL is to translate from logical sectors to physical sectors. The mapping between the logical address and the physical address can be managed at the sector, block, or hybrid level. The disadvantage of sector-level mapping is that the size of mapping table is too large to be viable in the current flash memory-packaging technology. The block-level mapping also has a serious pitfall: when an overwrite request for a logical sector is necessary, the corresponding block is remapped to a free physical block. In the hybrid scheme, in addition to a block-level mapping table, a sector-level mapping table for a limited number of blocks is maintained. Thus, it satisfies the size limitations of mapping information and also mitigates the remapped problem drastically [4].

Hybrid mapping scheme is known as log block

scheme. The key idea of log scheme is to maintain a small number of log blocks in flash memory to serve as write buffer blocks for overwrite operations. When an overwrite operation occurs with the same logical page data, the incoming data is written to a free page and the previous data becomes invalid. There are three kinds of approaches depending on the block association policy, block-associative sector translation (BAST) [6], fully-associative sector translation (FAST) [3] and set-associative sector translation (SAST) [5]. In the BAST scheme, a log block is used for only data block. In the FAST scheme, a log block can be used for several data blocks. In the SAST scheme, a set of log blocks can be used for a set of data blocks. Generally, a round robin policy is used in selecting a victim log block, but all these existing schemes have no consideration on the buffer.

2.3 Flash Buffer Cache

There are not many researches on the buffer cache in flash memory systems. Clean first LRU (CFLRU) [7] is a buffer cache management algorithm for flash storage. It attempts to choose a clean page as a victim rather than dirty pages because writing cost is much more expensive. The flash aware buffer policy (FAB) [9] is another buffer cache management policy used for flash memory. In FAB, the buffers that belong to the same erasable block of flash memory are grouped together. When the buffer is full, all pages in the same block are evicted to flash memory; this algorithm reduces the garbage collection cost. BPLRU [8] is another buffer management scheme which especially improves the performance of random writes. This scheme also uses the block-level buffer replacement like FAB, but it allocates and manages buffer memory only for write requests. These schemes handled only the buffer cache management, but not consider the garbage collection and did not refer to the FTL mechanism.

3. MOTIVATION

In this section, we will explain the motivation and benefits of our approach. When the garbage collector merges blocks to reclaim them, a large number of pages should be migrated, i.e. read dirty pages and write to new blocks, but it is possible that a few of these pages has been overwritten in a block-level buffer before they are migrated. Consider an example described below:

Figure 1 is a block-level buffer, when there is no free space in buffer, the least recently used block is selected as a victim block, and all sectors in the victim block are flushed to flash memory. In this example, block 3 is selected as the victim block, and sectors 12 and 15 are flushed. Next will be block 1, sectors 5 and 6 will be flushed to memory in the shortly future.

Figure 2 shows an example of merging blocks in the FAST scheme [3]. B0 and B1 are data blocks,

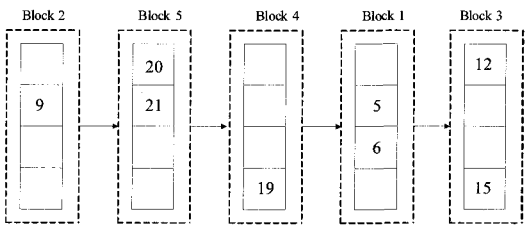


Fig. 1. Block-level buffer

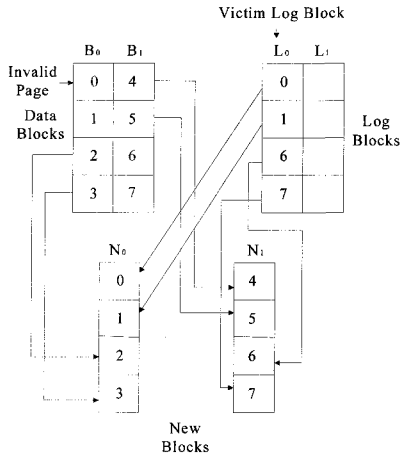


Fig. 2. A process of merging blocks

L0 is a log block. Each block consists of four pages. When new page update comes, the new page will be written on log blocks corresponding to data blocks if there is no empty space in data blocks, and then the original pages in data blocks are set as invalidated. As show, pages 0, 1, 6 and 7 in data blocks B0 and B1 are set as invalidated. If the garbage collector selects the log block L0 as a victim block, new data blocks N0 and N1 are allocated and valid pages in B0, B1 and L0 are moved to N0 and N1. After the page migration, blocks B0, B1 and L0 are erased.

When merging log block and data blocks in flash memory, if we can examine the block-level buffer, we find that moving sector 5 and 6 to new data block is unnecessary because they will be invalidated shortly when the block 1 in buffer is evicted to flash memory. If we can move sector 5 and 6 from buffer instead of B1 and L0, we can avoid lots of potential page migration.

4. BUFFER-AWARE BLOCK MERGE

In order to avoid potential page migrations, we should refer to the content of buffer during the block merge. If we find the corresponding block in the buffer when we do merge operation, it may be a good idea to flush it to flash memory in advance. But there exists a trade-off between merge performance and the buffer hit ratio. When hot pages are selected to flush into flash memory, the buffer hit ratio will decrease. In this situation, it is beneficial to choose another log block as victim log block. In this section, we first introduce an approach focus on how does block-level buffer flush data to flash memory, then we will propose a garbage collection method based on block-level buffer, finally we discuss the victim block selection technique.

4.1 Block-level buffer to flash memory

When the buffer is full that can not allocate any space to the new coming pages, the least recently used block is selected as a victim block, and all the sectors in the victim block are flushed from the buffer to flash memory. This is the basic policy used in block-level buffer management.

In the general FAST scheme, if an individual logical sector write operation comes from the file system, the FAST will examine whether the corresponding physical sector is empty or not. If it is empty, the sector is written to the corresponding data block, otherwise, the sector is written to the log blocks which shared to any data block.

When the file system uses the block-level buffer to manage the write operations, the write requests will be executed block by block. In other words, all the sectors belong to the same block which resides at the end of the buffer list should be written. In this case, the write requests consist of several write operations. As an example of figure 1, when block 3 is selected as the victim block, sector 12 and 15 are written to flash memory at the same time.

Above all, combining the FAST scheme in flash memory and block-level buffer in file system, a write request need one or several sector write operations.

4.2 Buffer-aware block merge

If no more empty sectors exist in the log blocks, the proposed garbage collection approach chooses one of the log blocks as victim and merges the victim block with its corresponding data blocks and the corresponding buffer blocks. The victim selection will be discussed in next section.

The merge operation proceeds as follows:

First, given a log block as victim block, find the corresponding data blocks and allocate the same amount of free blocks.

Second, search the buffer to find whether there exists blocks which have the same number to the corresponding data blocks or not.

Third, flush the sectors in buffer blocks to free blocks, and copy the most up-to-data version from

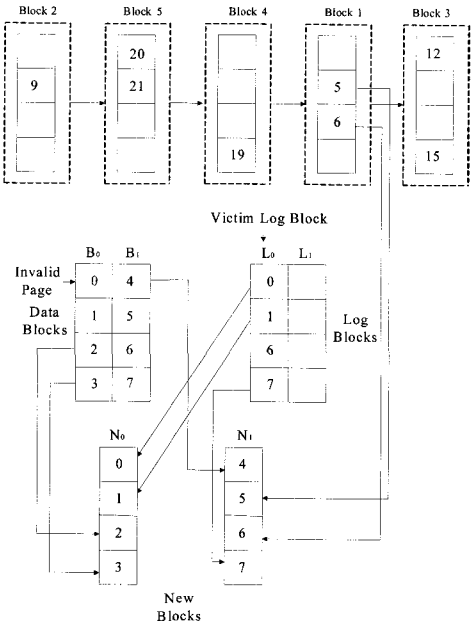


Fig. 3. Merge blocks refer to contents of buffer

the log blocks to free blocks, then fills each empty sector in the free block with its corresponding sector in the data blocks.

Fourth, erase the log block and corresponding data blocks.

Consider an example as shown in figure 3. When log block L0 is selected as victim block, we find that B0 and B1 are its corresponding data block and there also exists block 1 in the buffer. Next, sector 5 and 6 in buffer are first flushed into new blocks, then copy sector 0, 1 and7 in log block to new blocks, finally fill the empty sectors using sector 2, 3 and 4 in data blocks. After merge all the blocks, erase log block L0 and data blocks B0 and B1. Figure 4 shows the buffer aware merge algorithm.

4.3 Victim block selection

In step two of the proposed approach, there may cause three situations according whether the blocks which have the same number to corresponding data blocks in buffer is found or not.

1): Not found. In this case, the garbage collection

```
Input:
LogBlock: Victim log block
Output:
Return: Block erase numbers and page copy numbers
Value:
pNumLogBlock: Page numbers in LogBlock
dataBlockVector: Store data block number
sizeDBV: Size of data block vector
newBlock: New block for merging
mergeBlockAwareBuffer( LogBlock )
Begin
01: for   i = 0  to pNumLogBlock
02: if page in LogBlock is valid
03: Store page to dataBlockVector
04: end if
05: end for
06: Allocate new block newBlock
07: for   j = 0 to sizeDBV
08: searchBuffer()
09: if searchBuffer() is true
10: write block from buffer to newBlock
11: remove block from buffer
12: end if
13: if page in data block is valid
14: write page from data block to newBlock
15: end if
16: if page in log block is valid
17: write page from log block to newBlock
18: end if
19: erase data block
20: erase log block
End
```

Fig. 4 Buffer aware merge algorithm

only need merge the log block and its corresponding data blocks. This process is done like the buffer-unaware merge operation.

2): Found and the found blocks belong to the least recently used blocks. If the blocks locate at the near end of the buffer list, that means these blocks will be flushed to flash memory in the shortly future. It is appropriate to move these sectors in the buffer into the new free blocks.

3): Found but the found blocks are hot blocks. In this case, the blocks locate at the near start of the buffer list, which means these sectors in the found blocks will be updated frequently. If we choose these blocks to flush to flash memory, the buffer hit ratio will decrease.

Therefore, it is beneficial to find another log

block as victim log block.

Here we use a novel concept named locality probability to judge the importance of the block in the buffer. The idea of locality probability is described as follows:

We assume the current number of blocks in buffer is n, the basic locality probability of the block which reside in the end of the buffer list is p, which is the lowest value, the difference value between two blocks is x, this means the locality probability of the front block is larger than its back block, and their difference value is x, the total sum of all block locality probability is 1. So we get:

$$P + (p+x) + (p+2x) + \ldots\ldots + (p+(n-1)x) = 1$$

If we set the difference value x a specific value, according to the number n, we can calculate the basic locality probability p and the locality probability of every block. In the real world, set a threshold of the locality probability depending on specific devices. The blocks whose locality probability larger than the threshold considered as hot blocks, and less than the threshold considered as least recently used blocks.

5. EXPERIMENTS

5.1 Experimental environments

In our experiments, we simulate the flash memory and buffer cache, then comparing the results of our method and the others which collect garbage don't refer to the contents of buffer. As mentioned above, the proposed method will generate the block erase number and page copy number, so we get this information as the results to compare the performance.

All the experiments are done in windows XP operating system and based on Microsoft visual studio 2005.

5.2 Experimental results

We assume that every block consists of 4 pages, the flash memory has 100 blocks, and the log block group has 10 log blocks, the buffer can maintain 10 blocks when it is full. The data are inputted randomly. We input data by the amount of 100 pages, 200 page, 300 pages, 350 pages and 390 pages, the results are shown as figure 5 and figure 6.

In figure 5 , X-axis denotes the random input page number for write to flash memory and Y-axis represents the numbers when erasing blocks and copying pages. Since the input is random, when the input page number is 100 which is more less than the total page number of flash memory, there need not any merge operation. From the figure, we find that the proposed method can reduce more block erase numbers and page copy numbers as the input page numbers increase. Especially, the proposed method reduces page copy number more than block erase number.

In figure 6, the line marked diamond denotes the page copy number by buffer unaware method, and line marked circle represents page copy number by our proposed method, marked rectangle line denotes

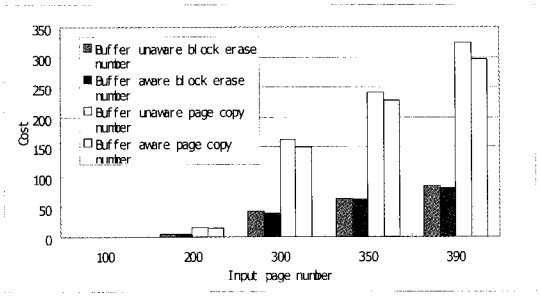


Fig. 5. Simulation results

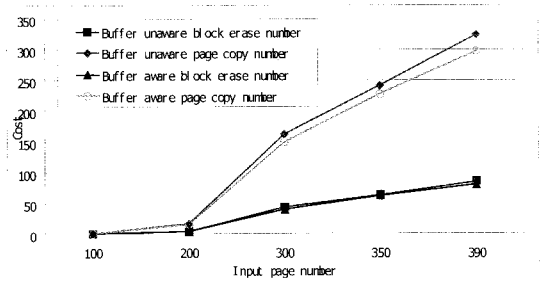


Fig. 6. Compare with two kinds approach

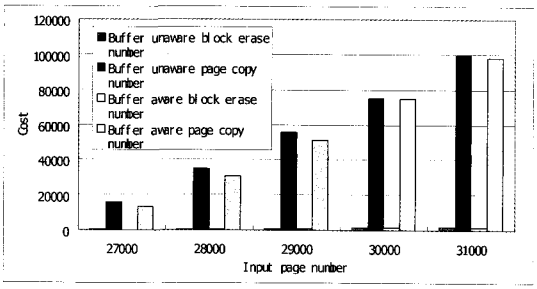


Fig. 7 The results of increasing input page number

buffer unaware block erase number, and marked triangle line represents buffer aware block erase number. From the figure, it is clear that the proposed method reduces the costs compared buffer unaware buffer method.

Next we simulate a real size flash memory whose capacity is 16MB. The results are shown as figure 7.

We assume that in the 16MB flash memory, the size of page is 512KB, a block consists of 64 pages, log block size is 128, and buffer length is 32.

When the flash memory is not very full, the proposed method can save more unnecessary costs. When the flash memory reach full nearly, the saved costs fall down because the flash cannot accommodate any more pages .In generally, the proposed method can reduce block erase number and page copy number by 4.6% and 4.7%. On the other word, the performance of flash memory is improved 4.6%.

6. CONCLUSION AND FUTURE WORK

We have presented a block-level buffer aware garbage collection technique which searches the contents of buffer when merging blocks to reduce the garbage collection cost. This method can improve the efficiency of the block merge by reducing the potential unnecessary block erase numbers and page copy numbers. Our experiments show that the proposed method can improve the performance up to 4.6%. The victim block selection approach improves the I/O performance but does not decrease the buffer hit ratio by selecting a proper block as victim block using the concept of locality probability.

In the future work, we will give an explicit analysis about locality probability to make the best performance between buffer hit ratio and the flash blocks merge cost.

REFERENCES

[1] D.B. Lomet, "Bulletin of the Technical Committee on Data Engineering," IEEE Computer Society, Vol. 30, No. 3, Sep. 2007.

[2] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," ACM Computing Surveys, 2005.

[3] S.W. Lee, D.J. Park, T.S. Chung, 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, No.3, 2007.

[4] C. Park, W.M. Cheon, J.G. Kang, K.G. Roh, W.H. Cho and J.S. Kim, "A reconfigurable FTL architecture for NAND flash-based applications," ACM Transactions on Embedded Computing Systems, Vol. 7, No. 4, 2008.

[5] J.U. Kang, H.S. Jo, J.S. Kim, and J.W. Lee, "A super block-based flash translation layer for NAND flash memory," International Conference On Embedded Software, pp. 161-170, 2006.

[6] J. Kang, 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, No.2, pp. 366-375, 2006.

[7] S.Y. Park, D.W. Jung, J.U. Kang, J.S. Kimand J.W. Lee, "CFLRU: a replacement algorithm for flash memory," International Conference on Compilers, Architecture and Synthesis for Embedded Systems, pp. 234-241, 2006.

[8] H. Kim, and S.J. Ahn, "BPLRU: a buffer man-

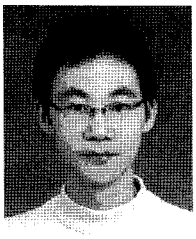
agement scheme for improving random writes in flash storage," Proceedings of the 6th USENIX Conference on File and Storage Technologies, 2008.

[9] H. Jo, J.U. Kang, J.S. Kim, and J. Lee, "FAB: Flash-aware buffer management policy for portable media players," IEEE Transactions on Consumer Electronics, Vol. 52, No.2, pp. 485-493, 2006.

[10] L.P. Chang and T.W. Kuo, "An efficient management scheme for large-scale flash-memory storage systems," Symposium on Applied Computing, pages 862-868, 2004.

[11] K.H. Parkand S.H. Lim, "An Efficient NAND flash file system for flash memory storage," IEEE Transactions on Computers, Vol. 55, pages 906-912, 2006.

[12] Intel Corporation, ,"Understanding the Flash Translation Layer (FTL) Specification," White Paper, http://www.embeddedfreebsd.org/Documents/Intel-FTL.pdf, 1998.

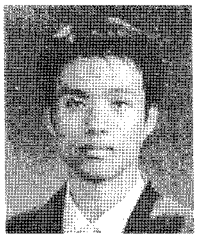


Liangbo Li

2003. 9~2007. 7 Dept. of Computer Science and technology, ChongQing Univ. of Post and Telecom. China.(B.S)

2008. 2~present Dept. of Computer Science and Engineering, .Inha Univ.(MS)

Flash Memory, Database, Data Stream Management.



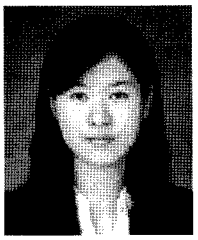
Song-Sun Shin

2006 Dept. of Computer Education, Seowon Univ.(B.S)

2008 Dept. of Computer Science and Engineering, Inha Univ.(M.S)

2008~Present Dept. of Information Technology, Inha Univ. (Ph.D)

Spatial Database, Grid Database, LBS, u-GIS, Data Stream Management System



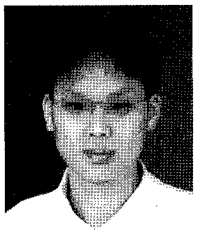
Yan Li

2002 Dept. of GIS Engineering, Univ. of Post and Telecom. China(B.S)

2006 Dept. of Computer Science and Engineering, Inha Univ.(M.S)

2008~Present Dept. of Information Technology, Inha Univ.(Ph.D)

Spatial Database, Spatial Data Warehouse, GIS, USN, Data Stream Management System



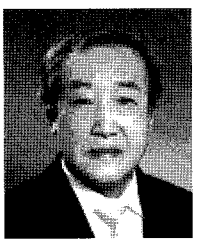
Sung-Ha Baek

2001 Dept. of Computer Technology (B.S)

2005 Dept. of Computer Science and Engineering, Inha Univ.(M.S)

2007~Present Dept. of Information Technology, Inha. (Ph.D)

Data Stream Management, Cluster System, LBS



Hae-Young Bae

1974 Dept. of Applied Physics, Inha University(B.S)

1978 Dept. of Computer Science and Engineering, Yonsei Univercity.(M.S)

1989 Dept. of Computer Science and Engineering, Soongsil University.(Ph.D)

1985 Guest Professor, Univ. of Houston

1982~Present, Professor, Dept. of Computer Science and Information Engineering, Inha University

1999~Present, Director, Intelligent GIS Research Center

2000~Present, Emeritus Professor, Chongqing University of Posts and Telecommunication, China

2006~2009 Director, Graduate School, Inha University.

Distributed database, Spatial database, Geometric information system, Multimedia database