논문 2009-46SD-3-6

NAND 플래시 파일 시스템을 위한 내용기반 블록관리기법을 이용한 마운트 시간 감소와 지움 정책

(Mounting Time Reduction and Clean Policy using Content-Based Block Management for NAND Flash File System)

조 원 희*, 이 동 환*, 김 덕 환*

(Won-Hee Cho, Dong-Hwan Lee, and Deok-Hwan Kim)

요 약

플래시 메모리는 비휘발성, 저전력, 빠른 입출력, 충격에 강함 등과 같은 많은 장점을 가지고 있으며, 모바일 기기에서의 저장 매체로 자주 사용이 증가 되고 있다. 이에 따라 임베디드 디바이스에 널리 사용되는 NAND 플래시 전용 파일시스템인 YAFFS에 관한 연구가 활발히 이루어지고 있다. 하지만 기존의 YAFFS는 마운트 시 모든 페이지의 스페어 영역을 스캔함으로써 마운트 속도가 상당히 오래 걸리며, 기존의 지움 정책에서 플래시메모리의 특성인 마모도 제한을 고려하지 않은 지움 정책(Garbage-Collection)을 사용하는 문제점을 가지고 있다. 따라서 본 논문에서는 YAFFS의 마운트 과정에서의 문제점을 해결하기 위해 블록을 내용기반 리스트로 관리하고 마운트 할 때 일부 스페어 영역만을 읽어 기존의 마운트 시간을 감소시키는 기법을 제시한다. 또한 기존의 마모도 기법의 문제점을 해결하기 위해 내용기반 지움 정책을 사용하는 블록 스왑기법을 제안 한다. 실험에서는 파일의 크기를 다양하게 분류하여 기존의 파일시스템들과 비교하였다. 내용기반 YAFFS가 JFFS2보다는 82.2% 기존의 YAFFS보다는 42.9%의 마운트 평균시간이 감소하였으며, 기존의 지움 정책과 비교하여 추가적인 삭제나 지움 횟수가 없으며 제안한 블록 스왑기법은 마모도를 균일화하여 약 35%의 수명 증가를 보여준다.

Abstract

The flash memory has many advantages such as low power consumption, strong shock resistance, fast I/O and non-volatility. And it is increasingly used in the mobile storage device. Many researchers are studying the YAFFS, NAND flash file system, which is widely used in the embedded device. However, the existing YAFFS has two problems. First, it takes long time to mount the YAFFS file system because it scans whole spare areas in all pages. Second, the cleaning policy of the YAFFS does not consider the wear-leveling so that it cannot guarantee the duration of data completely. In order to solve these problems, this paper proposes a new content-based YAFFS that consists of a mounting time reduction technique and a content-cleaning policy by using content-based block management. The proposed method only scans partial spare areas of some special pages and provides the block swapping which enables the wear-leveling of data blocks. We performed experiments to compare the performance of the proposed method with those of the JFFS2 system and YAFFS system. Experimental results show that the proposed method reduces the average mounting time by 82.2% comparing with JFFS2 and 42.9% comparing with YAFFS. Besides, it increases the life time of the flash memory by 35% comparing with the existing YAFFS whereas no overhead is added.

Keywords: NAND Flash Memory, Mounting Time, Clean Policy, YAFFS, JFFS2

* 정회원, 인하대학교 전자공학과
(Department of Electronic Engineering, Inha University)

※ 본 논문은 정보통신부 출연금으로 ETRI, SOC산업 진흥센터에서 수행한 IT SOC 핵심설계인력양성사업의 연구결과입니다.

※ 본 연구는 지식경제부와 한국산업기술 재단의 전략기술인력양성사업으로 수행된 연구결과임.

접수일자: 2008년12월16일, 수정완료일: 2009년2월3일

Ⅰ. 서 론

PDA, PMP와 RFID 리더 등의 모바일 기기에서의 저장매체로 플래시 메모리 사용이 증가하고 있다[8]. 플래시 메모리는 비휘발성 특징을 가지고 있으며, 하드디스크에 비해 견고하다. 저 전력으로 동작이 가능하며 접근 시간이 하드디스크 보다 빠르다. 또한 크기가 작아 휴대 기기에 적합하다. 그러나 단점으로 하드 디스크에 비해 가격이 5~10배정도 비싸며, 이미 데이터가 있는 공간에 새로운 데이터를 쓰고자 할 때는 지움 과정을 수행한 다음에서야 데이터를 저장 할 수 있다. 또한 읽는 속도는 매우 빠르지만, 쓰기 속도와 지우는 속도가 상대적으로 느리고, 한 번에 지울 수 있는 크기가 일정하며, 상온에서 지울 수 있는 회수가 정해져 있다[2]. 또한 이런 플래시 메모리는 NOR 형태와 NAND 형태의 플래시 메모리 형태로 구분 할 수 있다. NOR 형태의 플래시 메모리는 빠른 읽기 속도의 장점을 가지며, NAND형태의 플래시 메모리는 NOR 형태의 플래시 메모리 보다 값이 싸고, 대용량 임베디드 장치에 사용하기에 용이하다는 장점을 가지고 있다. 표 1에서는 플래시 메모리의 기본 연산 수행시간을 보여준다[7].

그림 1은 NAND 플래시의 구조를 보여 준다. 플래시 메모리는 블록들로 구성되며 각블록들은 페이지들로 구성 된다. 작은 블록의 경우 블록은 32개의 페이지로 구성되며 각 페이지는 512 bytes의 크기를 가진다. 반면 큰 블록의 경우에는 64개의 페이지로 구성 되며 각 페이지는 2048bytes의 크기를 가진다. 또한 플래시 메모리 안에 자유공간이 부족할 시 블록 안에 무효화 된 페이지들을 정리하여 자유공간을 할당하는 가비지 컬렉션 작업이 이루어진다.

표 1. 플래시 메모리 기본 연산 수행시간
Table 1. Execution time of basic flash memory operations.

Media	Read	Write	Erase
DRAM	60ns (2B) 2.56us (512B)	60ns (2B) 2.56us (512B)	N/A
NOR Flash	150ns (1B) 14.4us (512B)	211ns (2B) 3.53us (512B)	1.2s (128KB)
NAND Flash	10.2us (1B) 35.9us (512B)	201us (2B) 226us (512B)	2ms(16KB)
Disk	12.4ms (512B) (average)	12.4ms (512B) (average)	N/A

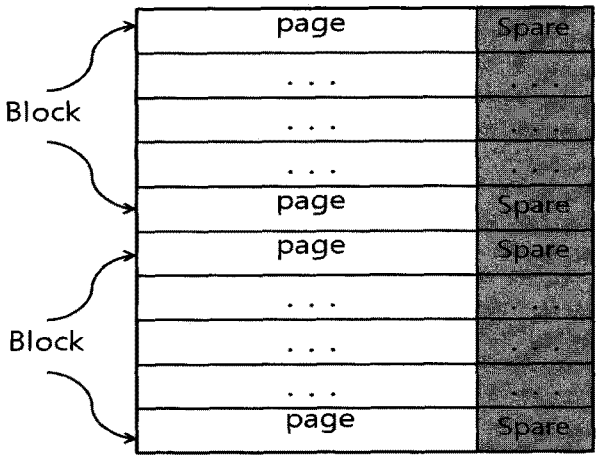


그림 1. 플래시 메모리 구조
Fig. 1. Flash memory structure.

앞서 언급한 플래시 메모리의 특성으로 인하여 기존 파일시스템을 플래시 메모리에 바로 적용할 수 없으므로 NAND플래시 전용 파일 시스템에 대한 연구가 활발히 진행되어 왔다. 대표적인 예로 FTL (Flash Translation Layer), JFFS2 (Journaling Flash FileSystem) YAFFS (Yet Another Flash FileSystem) 등이 있다[9~11]. FTL은 순차적인 플래시 공간이 디스크의 섹터처럼 보이도록 하기 위해 매핑 (mapping) 관리를 수행하는 드라이버 형식으로 구현되어 있다. JFFS2는 플래시 공간을 순차적으로 저장하는 LFS (Log-structured File System) 처럼 플래시 메모리에 대한 갱신 연산을 추가 연산으로 변형하여 처리하며, 이를 통해서 플래시 메모리의 덮어쓰기가 허용되지 않는 문제를 해결하였다[12]. 현재YAFFS의 기본적인 지움 정책은 플래시 메모리의 특성인 쓰기 횟수제한의 마모도 문제 해결과 Garbage-Collection 해결방법이 미비하다.

본 논문에서는 기존의 YAFFS의 문제점인 마운트속도를 개선하는 내용기반 블록 관리 기법을 제안하며, 효율적인 내용기반 Garbage-Collection과 내용 기반에 따른 블록 스왑을 통하여 마모도 관리를 향상 시킬 수 있는 지움 정책을 제안한다.

Ⅱ장에서는 플래시 메모리의 특성과 리눅스에서 사용하고 있는 NAND 플래시 전용파일 시스템인 YAFFS에 대하여 살펴보고, Ⅲ장에서는 YAFFS의 블록을 내용기반 리스트에 관리함으로써 기존의 마운트 속도를 향상시키는 기법과 블록스왑을 통해 마모도를 고려하는 내용기반 지움 정책을 제안한다. Ⅳ장에서는 기존

YAFFS와 JFFS2 등의 파일시스템과 마운트 성능 및 마모도를 비교하며, V장에서 결론을 맺는다.

II. 관련연구

1. JFFS2(Journaling Flash File System 2)

JFFS는 데이타를 로그 형태로 플래시 메모리에 순차적으로 쓰고, 읽기 연산은 로그를 역순으로 검색하여 가장 최신의 데이타를 읽어 들인다. JFFS에서는 저널링 노드로서 jffs_node라는 구조체를 사용하는데, 이것의 크기는 48Bytes로 상당히 크다. 이런 오버헤드를 줄이기 위해 JFFS2는 next_in_ino, next_phys, flash_offset, totlen 값만으로 구성된 jffs2_raw_node_ref라는 구조체 (16Bytes)로 jffs_node를 대체하여 메모리에 저장한다. 또한, 플래시 메모리의 공간을 효율적으로 활용하기 위해 데이터 압축 기능을 사용한다. 메모리 사용량이48Bytes에서 16 Bytes로 줄었다고는 하지만, 플래시 메모리가 128MB ytes일 때, 218페이지를 필요로 할 수 있기 때문에 jffs2 _raw_node_ref를 위한 공간만 222Bytes, 즉 4MBytes를 할당해야 하는 문제점이 있고, 노드를 찾고 파일의 구조를 결정하기 위한 스캔 시간이 길다. 또한, 마운트할 때 플래시 메모리 전체를 스캔해야 하기 때문에 128MBytes NAND의 경우 마운트에 25초가 걸리는 것으로 측정되었다[5]. 이와 같이 JFFS2는 NOR 플래시 메모리를 기반으로 설계되었기 때문에 NAND 플래시 메모리용 파일 시스템으로 사용되기에는 메모리 사용량, 마운트 및 플래시 메모리 스캔 시간 그리고 가비지 컬렉션 시간 등에서 여러 가지 문제점이 있다[5].

2. YAFFS(Yet Another Flash File System)

YAFFS는 JFFS2가 저널링에 사용되는 메모리 소모량이 큰 것과, 느린 마운팅 속도를 해결하기 위해 개발된 NAND 플래시 전용 파일 시스템이다.

YAFFS는 페이지의 크기가 512Byte인 플래시 메모리만이 사용가능하며 최대 파일 크기 512MB, 최대 파일 수 260,000, 최대 파일 시스템 크기 1GB인 제약을 가진다. YAFFS2의 경우 2Kbyte 페이지 플래시 메모리가 사용가능하며 최대 8GB 파일 시스템 크기를 가질 수 있다[3]. 플래시 메모리에서 읽기와 쓰기는 페이지 단위로 수행되며, 삭제는 블록 단위로 수행된다. YAFFS는 그림 2와 같이 파일 데이터를 하나의 페이지와 동일한 크기인 Chunk로 나누어 플래시 메모리에

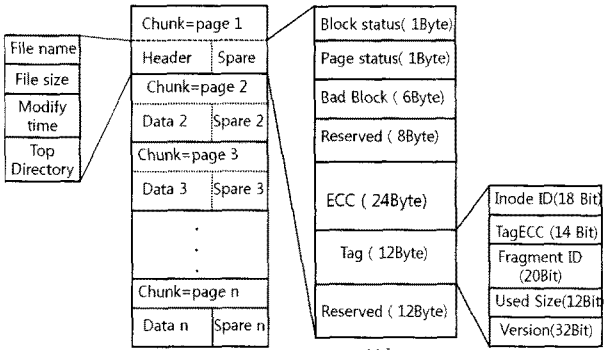


그림 2. YAFFS 플래시 메모리 구조
Fig. 2. YAFFS flash memory structure.

저장한다[4].

Chunk는 파일의 데이터와 스페어 영역으로 구분되며, 데이터 영역에 파일의 정보를 관리 하는 헤더 (Object Header) 가 저장된다. 헤더가 저장되는 경우는 스페어 영역의 ChunkID가 0으로 되어 있다. 이 헤더는 파일의 이름과 파일의 크기, 수정 시간, 상위디렉터리에 포인터 등으로 구성된다. ChukID가 0이 아니면 파일의 데이터라고 간주한다[13]. 또한 각 Chunk마다 스페어 영역이 존재한다. 스페어 영역에는 블록상태, 페이지상태, ECC영역 및 Tag가 존재한다. 페이지 상태 영역에는 유효(valid)/무효(invalid)페이지를 나타내는 정보가 있다. Tag영역은 페이지에 들어있는 데이터의 크기, 파일 업데이트 횟수 및 아이노드 식별자 등의 자료가 들어 있다.

파일 갱신과 파일 삭제 하는 경우에 그 파일이 들어 있는 페이지를 무효화하게 된다. 추후 가비지 컬렉션을 할때 무효화 된 페이지들을 포함한 블록을 삭제하게 된다.

YAFFS는 마운트 시에 플래시 메모리에서 스페어 영역만을 스캔하여 헤더가 저장되어 있는 페이지가 발견되면 그 페이지 데이터의 내용과 스페어영역에 저장된 파일상태를 읽어서 해당 파일의 오픈파일 테이블을 주 메모리에 동적으로 생성한다. 따라서 플래시 메모리의 전체 영역을 읽어야 하는 JFFS2에 비해 마운트시간이 훨씬 짧다. 그러나 마운트 과정에서 최소한 모든 페이지의 스페어 영역을 읽어야 하기 때문에 플래시 메모리의 크기가 커질수록 마운트 시간이 증가하게 된다.

2.1 YAFFS에서의 지움 정책

YAFFS에서의 지움정책 (Garbage-Collection)은 기본적으로 Greedy 정책을 사용하고 있다. Greedy정책의

사용으로 인해 수명감소(마모도) 문제가 발생하지만 YAFFS에서는 전체용량의 5%정도를 예약된(Reserved) 영역으로 할당하여 마모도가 한계에 도달하여 사용할 수 없는 블록을 예약된 영역에 저장된 블록으로 교체하는 방법을 사용하여 보완한다. 하지만 이는 근본적인 해결책이 아니다[3].

3. 지움정책(Clean-Policy)

플래시 메모리는 한번 사용한 블록을 다시 사용하기 위해서 항상 삭제작업을 거쳐야 한다. 삭제작업은 읽기, 쓰기 속도에 비해 지우는 속도가 상당히 느리며, 삭제작업은 블록단위로 이루어진다. 또한 마모도 제한이 존재하여 삭제 시 마모도를 고려하지 않으면 플래시 메모리 수명이 줄어들 수 있다. 이번 절에서는 기존의 지움 정책들을 소개한다.

3.1 Greedy 기법

Greedy 기법은 세그먼트 내에서 유효 페이지(valid page)가 가장 적은 블록을 지울 대상으로 선정하는 방식이다. 이 기법은 유효 페이지를 다른 블록으로 옮기는 비용이 적게 들고, 별도의 연산 작업이 필요하지 않다는 장점이 있다. 하지만, 시간적, 공간적으로 특정 세그먼트의 변경이 자주 발생되는 지역성 쓰기일 경우 비용이 커지며, 각 세그먼트의 지움 횟수를 균일하게 해주는 마모도 평준화(wear-leveling)가 전혀 고려되지 않는 단점이 있다[6].

3.2 Cost-Benefit 기법

Cost-Benefit 기법은 비용 (페이지 삭제비용+유효 페이지를 다른 곳으로 옮기는 비용)에 비해 이익이 많이 남는 세그먼트를 선택해서 삭제하는 지움 정책으로 식 1과 같이 비용대비 이익 비율을 이용한다. 단 비용은 삭제 및 이동에 필요한 시간이고 이익은 지움 과정 후 추가로 얻는 용량이다.

$$\frac{Benefit}{Cost} = \frac{Age \times (1-u)}{2u} \quad (1)$$

u는 해당 블록 내 유효 페이지의 비율이고 (1-u)는 무효도 페이지의 비율이다. 2u는 블록의 유효 페이지를 다른 블록으로 옮길 때 발생하는 읽고 쓰는 유효 페이지의 비율이다 Age는 블록에 페이지가 기록된 후 경과 시간이다. Cost-Benefit 기법에서는 Age를 이용하여 마모도 평준화를 추구하지만 이는 블록의 누적 지움 횟수가 아닌 블록에 기록된 시간이므로 근본적으로 지움 횟수에 대한 완벽한 마모도 평준화라고 볼 수 없다[6].

3.3 CAT 기법

CAT 기법은 식2와 같이 각 블록마다 값을 계산하여 그 값이 가장 작은 블록을 지움 정책의 대상으로 삼는 기법으로 블록 내 유효 페이지의 비율과 Age, 그리고 블록의 누적 지움 횟수를 이용한 기법이다.

$$\frac{u}{(1-u)} \times \frac{1}{Age} \times NmberOfCleaning \quad (2)$$

지움의 비용은 $\frac{u}{(1-u)}$로 표현이 되고, u와 (1-u)는 무효된 페이지에 대한 유효 페이지의 비율이다. Age는 블록이 기록된 이후의 경과 시간을 말하며, Age는 그림 2의 사전에 정의된 Age 변환 함수에 의해 결정 된다. 마지막으로, Number OF Cleaning은 블록이 지워진 누적 횟수를 말한다. 하지만 CAT기법은 사용자 쓰기 패턴이 순차 쓰기이거나 지역성이 있는 쓰기일 경우는 지움 횟수나 복사·횟수가 Greedy 기법과 Cost-Benefit 기법보다 적기 때문에 더 나은 성능을 보여주지만, 랜덤 쓰기일 경우에는 Greedy 기법 보다 지움 횟수와 복사 횟수가 더 많고 평균 처리량이 더 적어 효율성 측면에서 오히려 더 낮은 성능을 보여준다[6].

Ⅲ. 내용기반 YAFFS 파일시스템

플래시 메모리는 기존의 저장매체와는 달리 다음과 같은 특성이 있다. 첫째, 데이터 수정 시 본래 주소에 덮어 쓰기가 불가능하다. 즉, 쓰기 연산 전에 해당 메모리 공간이 초기화되어 있어야 한다. 둘째, 블록마다 지울 수 있는 횟수가 제한적이라는 단점을 가지고 있다[1]. 따라서 마운트 시간 증가와 마모도 횟수 제한을 극복하고 플래시 메모리를 효과적으로 사용하기 위해 다양한 지움 정책과 마모도 균등 사용 기법이 적용된 파일 시스템이 필요하다. 본 논문에서는 위의 단점들을 효율적으로 해결할 수 있는 내용기반 블록 관리를 통한 마운트 시간감소 기법과 내용기반 지움 정책을 제시하고자 한다.

1. 내용기반 블록관리 기법

YAFFS와 같은 파일 시스템에서는 각 블록 안에 파

일을 구분 할 수 있는 헤더가 저장되어 있기 때문에 한 블록 안에 포함된 파일 개수를 파악할 수 있다. 블록에서 파일 한 개당 하나의 헤더를 가지기 때문에 헤더의 개수를 통하여 블록 안에 들어있는 내용이 하나의 파일로만 이루어져 있는지 여러 파일이 혼합되어 들어있는지를 알 수 있다. 이를 기반으로 내용기반 YAFFS에서는 블록안의 파일 개수를 기반으로 블록을 세 가지(순수, 혼합, 자유)로 나눈다. 순수블록은 블록 안에 파일이 한 가지 내용의 파일로만 가득 차 이루어진 블록이며, 혼합블록은 두 가지 이상의 파일이나 자유페이지를 포함한 블록을 의미하며, 자유블록은 자유페이지만을 포함한 블록이다.

블록의 종류는 파일이 플래시 메모리에 쓰일 때 결정되며, 이는 다음과 같은 예를 통하여 확인 할 수 있다. 전체 용량이 512Mbyte인 NAND플래시 메모리에 한 블록의 크기가 128Kbyte, 한 페이지는 2Kbyte로 구성되어 있다고 가정한다. 이러한 플래시 메모리에 크기가 4.4Mbyte인 MP3파일이 쓰일 때 식3을 적용하면 35개의 순수블록이 생성되고, 1개의 혼합블록이 생성된다. P_n은 순수 블록의 개수이며, F_{size}는 파일의 크기 그리고 B_{size}는 블록의 크기를 나타낸다.

$$P_n = F_{size} / B_{size} \tag{3}$$

그림 3은 순수블록과 혼합블록이 어떻게 구성되어 있는지 나타낸다. 순수 블록과 혼합블록의 첫 번째 페이지의 데이터 영역에는 헤더가 들어간다. 헤더에는 파일에 대한 정보(메타 데이터)가 저장되어 있다. 또한 내용기반 블록관리를 위하여 블록의 첫 페이지 스페어 영역의 예약된 영역(Reserved)에 블록종류필드(Block Type Field) 정보를 추가한다. 이 블록종류필드에는 순수, 혼합, 자유 블록을 나타내는 정보가 포함되어 있다. 따라서 블록의 첫 페이지 스페어 영역을 스캔함으로써 블록안의 파일의 정보를 파악하여 마운트 할 수 있다.

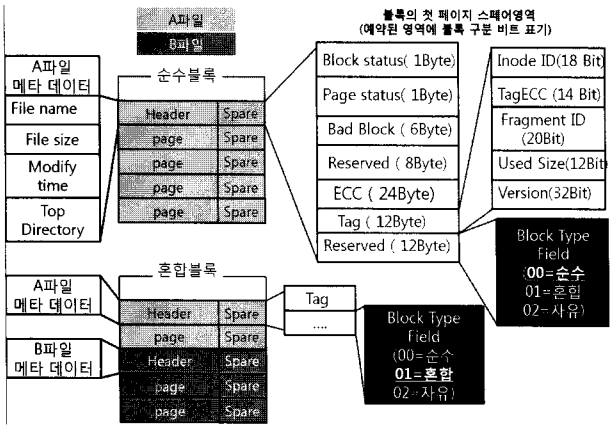


그림 3. 순수블록과 혼합블록의 구조
Fig. 3. The structure of mixed blocks and pure blocks

2. 내용기반 YAFFS의 마운트시간 감소

마운트 시에 해야 할 일은 기억장치에 저장되어 있는 일련의 데이터들을 의미 있는 자료로 만드는 과정을 거쳐야 한다. 플래시 메모리의 특성상 일정영역을 이를 위한 구간으로 분류해 둘 수 없기 때문에 마운트 시마다 매번 전체 블록의 첫 페이지의 데이터 영역에 저장된 헤더와 스페어영역에 저장된 파일정보를 통해 오픈파일 테이블을 구성한다[5]. 따라서 마운트 시간이 상당하다는 것을 알 수 있다. 본 논문에서는 이러한 단점을 극복하기 위해 마운트 시에 블록의 첫 페이지 스페어 영역에 블록종류필드를 추가함으로써 기존 파일시스템에 비해 마운트 시간을 감소하도록 하였다.

그림 4는 기존의 YAFFS 마운트 방법과 제안하는 내용기반 YAFFS의 마운트 방법을 나타낸다. 내용기반 YAFFS의 마운트과정에서 블록의 첫 페이지 스페어 영역의 블록종류필드를 통해 해당블록이 순수 블록인지 혼합블록인지를 판별하며, 순수블록일 경우에 블록 안에 쓰인 데이터는 같은 파일이므로 블록 안에 남은 페이지를 스캔할 필요 없이 첫 페이지만 스캔함으로써 마운트 시간을 (블록 안에 있는 전체 페이지 개수 -1) * 순수블록의 개수만큼 줄일 수 있다.

스캔 후에는 블록들을 각각의 리스트를 구성하여 내용기반으로 관리 한다. 그림 5는 각블록에 대한 마운트 방식을 알려주는 순서도이다.

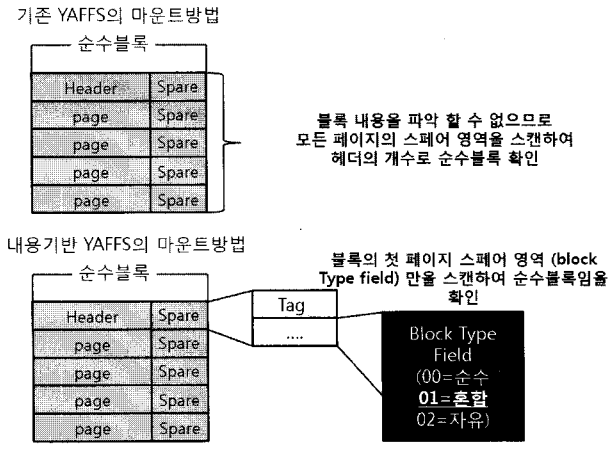


그림 4. 마운트방법 비교
Fig. 4. Comparison of the mounting method

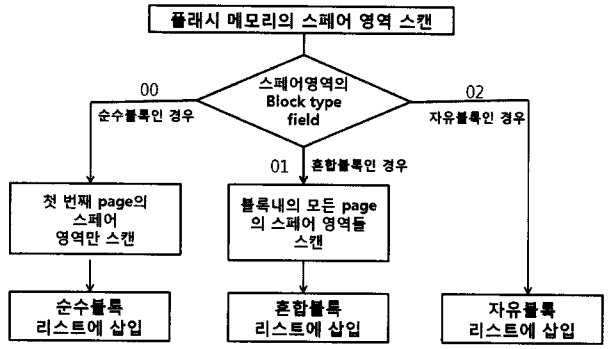


그림 5. 제안한 방법의 마운트 순서
Fig. 5. Mounting sequence of the proposed method.

3. 내용기반 지움 정책

3.1. 파일 삭제와 파일 쓰기

파일 쓰기나 파일 삭제 연산이 발생하면 해당 파일이 들어있는 페이지들이 무효화된다. 따라서 순수블록의 경우에는 동일한 파일의 내용을 담고 있으므로 파일 삭제 시 모든 페이지가 무효화 된다. 반면에 혼합 블록의 경우 해당 파일이 들어있는 페이지들만 무효화된다. 또한 파일 기록 시에 마모도 평준화 (wear-leveling)를 고려하여 기록한다[14].

3.2 내용기반 가비지 컬렉션

가비지 컬렉션은 자유블록 (가용공간)이 부족할 때 수행되며 무효화 페이지들로 이루어진 블록에 삭제 연산을 수행하여 블록을 초기화함으로써, 다시 새로운 데이터를 저장할 수 있는 자유 블록을 만드는 과정이다. 가비지 컬렉션은 모든 페이지가 무효화된 블록에 대해서 삭제 연산을 수행하는 것이 원칙이다. 그러나 자유블록이 일정한 임계 값 이하로 떨어지고 더 이상 모든 페이지가 무효화된 블록이 존재하지 않는 경우에는 일부 유효 데이터를 포함하고 있는 블록을 대상 블록으로 선정하여 유효 데이터를 다른 블록으로 복사한 후 무효화 된 블록을 삭제하는 방식으로 수행 된다. 제안한 내용기반 가비지 컬렉션 방법은 2단계로 구성된다.

1단계: 가용공간이 부족한 경우 자유블록 확보.
2단계: 블록 스왑을 통한 마모도 평준화.

3.1.1 자유블록 확보 (가용 공간이 부족한 경우)

그림 6은 내용기반 가비지 컬렉션의 수행 순서를 나타낸다.

데이터 기록 시 그 데이터가 차지하는 블록의 개수가

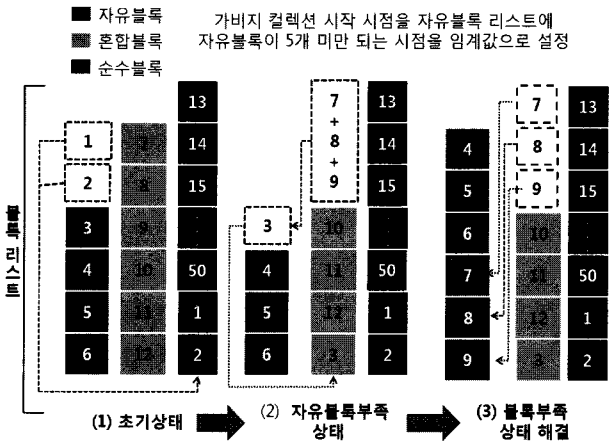


그림 6. 자유 블록 (가용공간) 확보 과정
Fig. 6. Allocation of free blocks.

2개라고 가정하면 그림 6의 (1)은 자유 블록 리스트의 블록 두개를 할당하며, 자유블록의 개수가 정해 놓은 임계값 (5개 블록)에 미달하게 된 상태를 보여준다. 임계값은 남아있는 최소의 자유블록 개수를 의미하며 전체 블록개수의 10%를 임계값으로 설정하였다. 임계값에 미달하면 자유블록 부족 상태가 되며, 이런 경우에는 혼합 블록들에 흩어져 있는 여러 데이터들을 하나의 혼합 블록으로 모아 자유블록을 확보할 수 있는지 확인한다. (2)는 혼합 블록들 중 유효페이지 수가 가장 적은 블록을 선택하여 자유블록에 옮겨주는 상태를 나타낸다. 예를 들면 혼합블록 (7,8,9)을 자유 블록 3에 모으면 2개의 자유블록이 확보되어 부족한 자유블록 문제를 해결할 수 있다. (3) 자유블록이 6개가 되어 블록부족 상태가 해결된 상황을 보여준다.

기존의 지움정책들은 모든 자유블록이 사용되어 부족하게 되면 무효화 된 페이지가 많은 블록을 선택하여 삭제한다. 따라서 긴 삭제시간을 소비하는 단점을 가지고 있다. 하지만 내용기반 지움정책은 자유블록의 임계값을 정해놓고 임계값 미만의 자유블록이 남았을 경우에는 혼합블록을 병합하여 임계값 수치 위로 상향시켜 긴 삭제 대기시간을 줄이는 삭제 블록 선택 방법을 사용하여 기존의 삭제 블록 선택방법의 문제점을 해결한다.

3.1.2 스왑을 통한 마모도 평준화 방법

내용기반 지움 정책에서는 순수 블록과 혼합 블록의 마모도 차이가 존재한다.

그 이유는 다음과 같다. 순수 블록은 한 가지 대용량 파일로만 이루어진 경우가 많다. 이런 대용량 파일 중

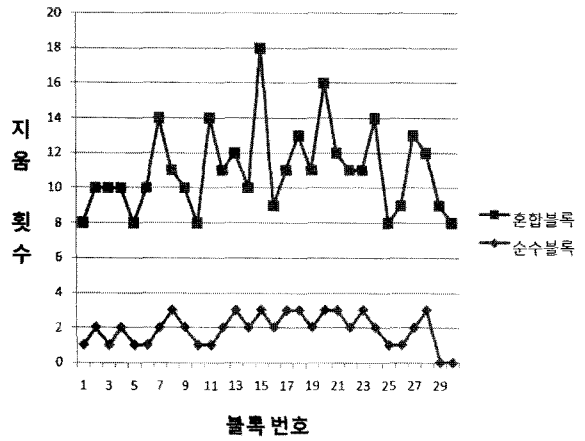


그림 7. 혼합블록과 순수블록 마모도 횟수 비교
Fig. 7. Comparison of wear-leveling between mixed blocks and pure blocks.

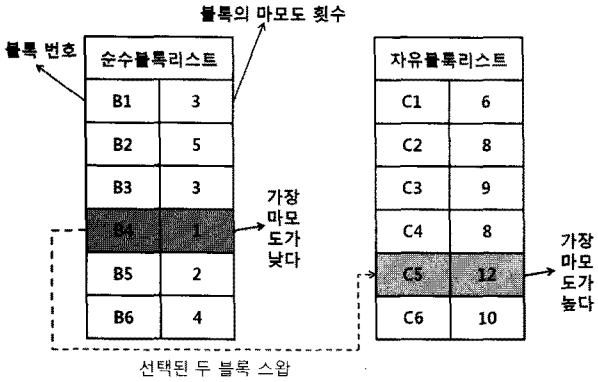


그림 8. 블록의 스왑과정
Fig. 8. The swapping process of two blocks.

한 부분이 갱신이 되면 이 부분의 페이지는 혼합블록의 빈 공간에 쓰이게 되며 이렇게 혼합블록에 갱신된 파일의 일부분의 페이지들은 재 갱신 시 또 다른 혼합블록에 쓰이게 된다. 그림 7은 순수블록보다 혼합블록의 마모도 횟수가 더 높음을 보여준다.

내용기반 가비지 컬렉션의 두 번째 단계로 블록 스왑을 통한 마모도 평준화 과정이 진행된다. 그림 8은 마모도 평준화를 위한 블록 스왑과정을 나타낸다.

먼저 순수블록리스트 중 마모도가 가장 작은 블록을 선택 한다. 그 후 자유블록 리스트 중 마모도가 가장 높은 블록을 선택한 후 두 블록의 마모도를 비교하여 차이가 일정 범위 (예:10)를 넘게 되면 두 블록을 스왑한다. 이 과정은 오버헤드이므로 유휴시간에 수행되어야 한다.

Ⅳ. 실 험

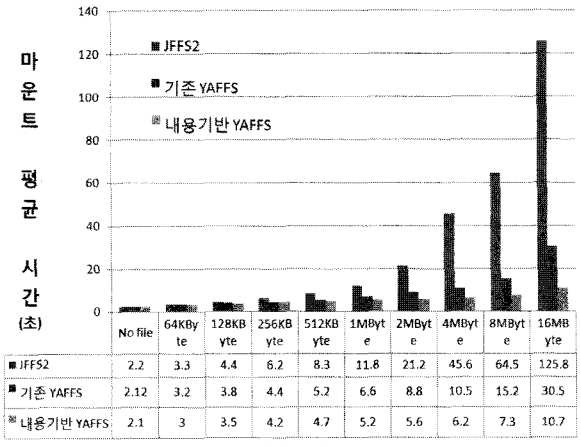
본 절에서는 다양한 실제 데이터를 이용하여 논문에서 제안한 내용기반 정책기반 마운트시간감소와 가비지 컬렉션시 마모도 관리를 통한 플래시 메모리의 수명 효율을 측정 하였다. 즉, 내용기반 YAFFS와 기존의 YAFFS및 JFFS2와 성능을 비교하였다. 이를 위해 Linux 2.6.17환경에서 다음과 같은 2가지 실험을 실행하였다.

첫 번째 실험은 JFFS2와 기존의 YAFFS 그리고 제안한 내용기반 YAFFS의 마운트 평균 시간을 비교하였다.

마운트 평균시간은 플래시 메모리에 있는 블록들의 스페어 영역을 스캔하여 주 메모리상에 파일정보를 구성하기까지 걸린 시간을 의미한다. 실험에 사용된 매개변수는 다음과 같다. 실험 데이터는 빈파일, 64Kbyte부터 16Mbyte까지 다양한 크기를 가지는 9개의 파일을 사용하였다.

그림 9에서는 JFFS2와 기존의 YAFFS 그리고 내용기반 YAFFS를 각각 마운트 시킨 후 각각의 마운트 평균 시간을 보여준다. 전체적으로 내용기반YAFFS가 JFFS2와 기존의YAFFS 보다 마운트 평균시간이 각각 82.2%, 42.9% 감소한 것을 보여준다. 또한 내용기반 YAFFS에서는 파일의 크기가 작은 경우보다는 파일의 크기가 큰 경우에 최대 91.7%, 67% 마운트 시간이 단축된 것을 확인할 수 있다.

두 번째 실험에서는 기존의 YAFFS와 내용기반 YAFFS의 지움정책들을 비교하기 위해 플래시 메모리



	No file	64KByte	128KByte	256KByte	512KByte	1MByte	2MByte	4MByte	8MByte	16MByte
JFFS2	2.2	3.3	4.4	6.2	8.3	11.8	21.2	45.6	64.5	125.8
기존 YAFFS	2.12	3.2	3.8	4.4	5.2	6.6	8.8	10.5	15.2	30.5
내용기반 YAFFS	2.1	3	3.5	4.2	4.7	5.2	5.6	6.2	7.3	10.7

그림 9. 마운트 시간 비교
Fig. 9. Comparison of mounting time.

의 수명 효율을 측정하였다. 실험에는 다음과 같은 시뮬레이션을 사용하였다.

- 마모도 측정 시뮬레이션: 블록 삭제 명령어 횟수측정, 다중 블록 삭제 명령어 횟수 측정, 지우기 요청된 블록 수 측정, 지우기 요청된 연속된 블록 수 측정 등의 4단계로 이루어진다[2].

- Postmark: 작은 크기의 파일들에 대한 쓰기, 읽기, 변경, 삭제 등의 트랜잭션을 수행한다[15]. 파일의 개수는 100개이며, 수행 할 트랜잭션은 각각 200, 500로 설정하였다.

그림 10, 11, 12에서는 Postmark를 이용하여 내용기반YAFFS의 지움 정책이 Greedy, Cost-Benefit, CAT 지움 정책들과 비교하여 기존기법 보다 오버헤더가 없음을 보여준다. 복사와 삭제 횟수에서 오버헤드가 없음을 보여 준다. 그림 10, 11, 12은 각각 순차쓰기, 지역성

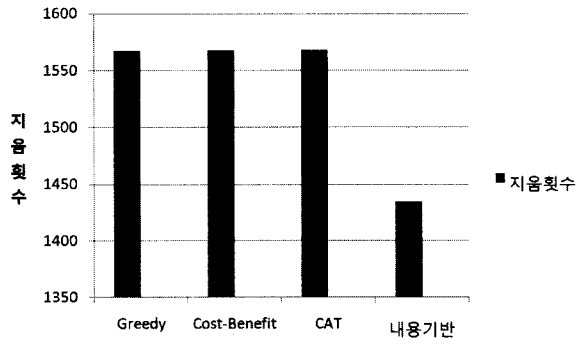


그림 10. 순차쓰기에서의 지움 횟수 비교
Fig. 10. Comparison of erase count for sequential writing

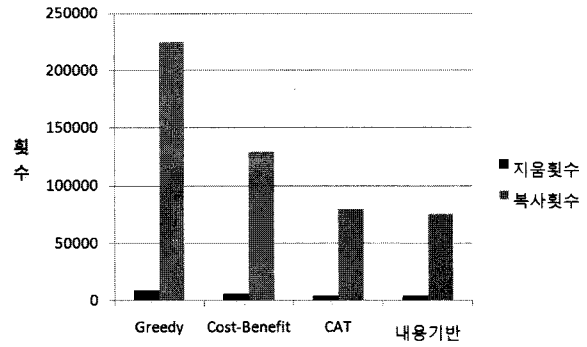


그림 11. 지역성 쓰기에서의 지움, 복사 횟수 비교
Fig. 11. Comparison of erase and move counts for local writing.

쓰기, 랜덤쓰기를 할 때 마모도를 나타낸다. 순차쓰기는 일반적인 자유블록을 할당하여 데이터가 쓰이는 것을 의미하며, 지역성 쓰기는 블록에 쓰인 데이터가 업데이트가 일어나 다른 블록에 쓰이는 경우를 의미한다. 그리고 랜덤쓰기는 순차쓰기와 지역성쓰기가 교대로 일어나는 경우를 의미한다.

그림 13, 14에서는 기존의YAFFS와 내용기반 지움 정책을 사용하는 내용기반YAFFS의 마모도 평준화에 대한 실험 결과를 보여준다. 기존의YAFFS는 블록들마다 마모도 횟수가 큰 차이를 보이며 일부 사용되는 블록만이 계속해서 사용되지만 내용기반 YAFFS는 블록들의 마모도가 균일하게 유지되는 것을 알 수 있다. Postmark 시뮬레이션에서는 각각 트랜잭션을 200, 500으로 설정하여 파일 삭제 및 복사를 반복하였다.

그림 15에서는 내용기반 지움 정책이 기존의 지움 정

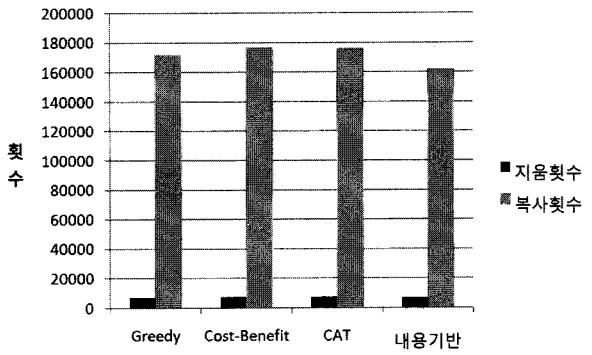


그림 12. 랜덤쓰기에서의 지움, 복사 횟수 비교
Fig. 12. Comparison of erase and move counts for random writing.

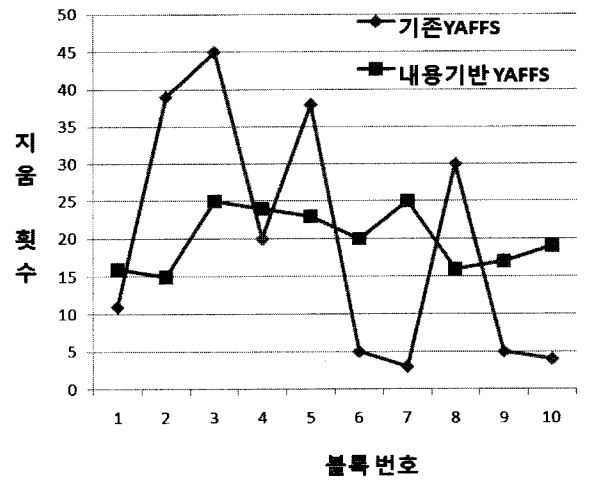


그림 13. 200회의 트랜잭션 반복 시 마모도 평준화 비교
Fig. 13. Comparison of wear-leveling when 200 transactions are repeated.

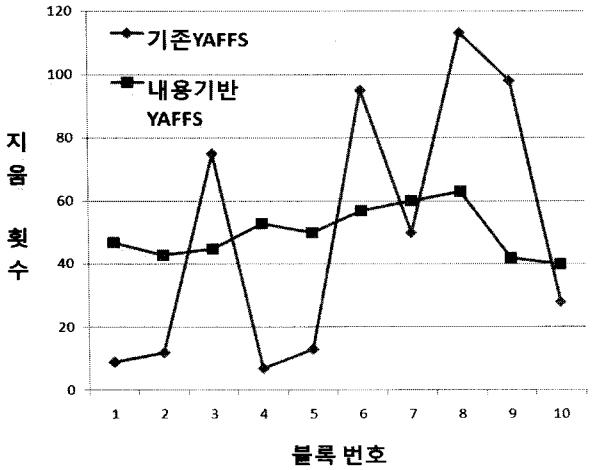


그림 14. 500회의 트랜잭션 반복 시 마모도 평준화 비교
Fig. 14. Comparison of wear-leveling when 500 transactions are repeated

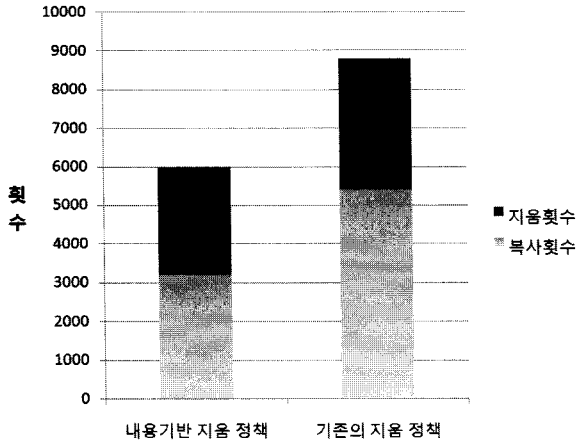


그림 15. 내용기반기법지움 정책과 기존의 지움 정책의 복사 및 지움횟수 비교
Fig. 15. Comparison of content-based clean policy and YAFFS clean policy in terms of copy and erase counts.

책보다 지움 횟수나 복사 횟수가 32%감소된 것을 보여준다. 내용기반 블록은 스왑과정에서 자유블록과 순수블록의 최댓값과 최솟값을 비교하여 스왑 한다. 하지만 기존의 지움 정책에서 스왑정책을 사용할 경우 블록의 구분 없이 마모도의 최댓값과 최솟값만을 비교한다면 스왑대상이 되는 블록에 유효한 데이터가 쓰일 수 있다. 이럴 경우 추가적인 새로운 블록을 임시블록으로 설정하여 스왑과정에서 사용하기 때문에 추가적인 복사 및 삭제 횟수가 발생하게 된다. 따라서 내용기반 지움 정책에서는 자유블록과 순수블록에서 마모도 횟수를 비교하여 스왑정책을 사용하여 추가적인 오버헤드를 방지한다.

V. 결 론

본 논문에서는 기존의 YAFFS보다 향상된 마운트속도와 마모도 평준화를 제공하는 내용기반 YAFFS 파일시스템을 구현하였다. 제안된 내용기반 YAFFS는 블록을 내용기반 리스트에 관리함으로써 마운트 시 일부 스페어 영역만을 읽어 마운트 시간을 감소시키고, 내용기반 리스트를 이용한 블록 스왑기법을 통하여 마모도를 효율적으로 관리하는 장점을 가지고 있다. 따라서 기존의 YAFFS가 적용된 플래시 메모리보다 빠른 마운트 속도를 제공하며, 마모도 평준화 기법을 통한 플래시메모리의 수명을 증가시킨다. 향후 내용기반 기법의 특징인 대용량 파일에서 성능이 향상되는 기술을 플래시 메모리 뿐만 아니라 다른 저장장치의 파일시스템에 적용하여 발전시킬 수 있는 연구를 수행할 예정이다.

참 고 문 헌

[1] 민용기, 박승규, "이동컴퓨터를 위한 플래시 메모리 클리닝 정책," 한국토인학회논문지, Vol.24, No.5A, pp.657-666, 1999.
[2] 이동환, 조원희, 김덕환 "다중 블록 지우기 기능을 적용한 퓨전 플래시 메모리의 FTL 성능 측정 도구 설계 및 구현," 대한전자공학회, 제31권, 제1호, pp.647-648, 2008.
[3] 이태훈, 박송화, 정기동 "YAFFS를 위한 파일 연산 최적화 기법," 정보과학회논문지, 제34권, 제1호 (B), pp.401-405, 2007.
[4] 한대만, 김성범, 구용완 "플래시 메모리를 위한 파일 시스템의 개선 방안," 한국인터넷정보학회, 제6권, 제2호, pp.733-736, 2005.
[5] 박상오, 김성조 "NAND플래시 메모리 기반 파일 시스템을 위한 빠른 마운트 및 안정성 기법," 정보과학회논문지, 제34권, 제11-12호, pp.683-695, 2007.
[6] 박광희, 김덕환 "휴대용 저장장치 시스템을 위한 Clustered Flash Translation Layer", 대한전자공학회, 제 45권, 제3호, pp.94-100, 2008.3
[7] 변시우 "F-Tree : 휴대용 정보기기를 위한 플래시 메모리 기반 색인 기법", 한국데이타베이스학회, 학술저널, 제13권, 제4호, pp.257-271
[8] F. Douglis, R. Caceres, F. Kaashoek, K. Li, B.Marsh, and J. A. Tauber, "Storage Alternatives for Mobile Computers," In Proceedings of the 1st Symposium on Operating Systems Design and Implementation, pp.25-37, 1994.

[9] Intel Coporation, "Understanding the Flash Transl ation Layer(FTL) specification". 1997.
[10] D. Woodhouse, "JFFS : The Journaling Flash File System", Technical Paper of RedHat inc. Oct. 2001.
[11] YAFFS Spec, http://www.aleph1.co.uk/yaffs/yaffs.html.
[12] M. Rosenblum and J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System", ACM Transaction on Computer System, Vol 10, No. 1, pp.26-52, 1992.
[13] Samsung Electronics Co., .NAND Flash Memory & SmartMedia Data Book., 2002.
[14] A. Kawaguchi, S. Nishioka, and H. Motoda, "A Flash-Memory Based File System", In Proceedings of Usenix Technical Conference, New Orleans, Louisiana , Jan. 1995, pp.155-164. 1995.
[15] Post mark, http://packages.debian.org/stable/utils/postmark

저 자 소 개

조 원 희(정회원)
2007년 명지대학교 정보통신공학부 통신공학과 학사 졸업.
2007년~현재 인하대학교 전자공학과 석사과정
<주관심분야 : 추천시스템, 시스템 소프트웨어>

김 덕 환(정회원)-교신저자
2003년 한국 과학 기술원 컴퓨터공학 박사.
2006년~현재 인하대학교 전자공학부 부교수
<주관심분야 : 시각정보처리, 스토리지 시스템, 임베디드 시스템>

이 동 환(정회원)
2008년 인하대학교 전자공학과 학사 졸업.
2008년~현재 인하대학교 전자공학과 석사과정
<주관심분야 : 임베디드시스템, 스토리지 시스템>