Journal of Korea Multimedia Society Vol. 11, No. 5, May 2008(pp. 651-660)

효율적이고 신뢰성 있는 플래시 파일시스템의 구현

진종원[†], 이태훈[††], 이승환[†††], 정기동[††††]

요 약

플래시 메모리는 비휘발성, 저전력, 빠른 입출력, 충격에 강함 등과 같은 많은 장점으로 임베디드 시스템에 많이 사용되고 있다. 그렇지만 제자리 덮어쓰기가 불가능하고 블록의 지움 횟수 제약이 있으며, 지움 연산을 페이지 단위로 해야 한다는 단점이 있다. 이러한 제약을 극복하기 위해 YAFFS와 RFFS 와 같은 NAND 플래시 파일 시스템이 제안되었다. YAFFS는 마운트 과정에서 필요한 데이터들을 얻기 위해 전체 플래시 메모리를 읽어야 하기에 마운트 시간이 느려진다. 따라서 빠른 마운트를 위해 필요한 모든 블록의 정보와 메타 데이터를 관리하는 RFFS 파일시스템이 제안되었다. 하지만 메타 데이터를 관리하기 위한 연산으로 인하여 파일시스템에 부담을 증가 시킨다. 본 논문에서는 위와 같은 문제점 해결을 위하여 최소한의 메타 데이터를 이용하여 빠른 부팅 및 복구를 제공 할 수 있는 NAND 플래시 파일 시스템을 제안한다. 그리고 실험을 통해 마운트 및 복구 성능 향상과 시스템에 부하가 없음을 보인다.

Implementation of Efficient and Reliable Flash File System

Jong-Won Jin[†], Tae-Hoon Lee[††], Seung-Hwan Lee[†††], Ki-Dong Chung[††††]

ABSTRACT

Flash memory is widely used in embedded systems because of its benefits such as non-volatile, shock resistant, and low power consumption. However, NAND flash memory suffers from out-place-update, limited erase cycles, and page based read/write operations. To solve these problems, YAFFS and RFFS, the flash memory file systems, are proposed. However YAFFS takes long time to mount the file system, because all the files are scattered all around flash memory. Thus YAFFS needs to fully scan the flash memory. To provide fast mounting, RFFS has been proposed. It stores all the block information, the addresses of block information and meta data to use them at mounting time. However additional operations for the meta data management are decreasing the performance of the system. This paper presents a new NAND flash file system called ERFFS (Efficient and Reliable Flash File System) which provides fast mounting and recovery with minimum mata data management. Based on the experimental results, ERFFS reduces the flash mount/recovery time and the file system overhead.

Key words: NAND flash(낸드플래시), Flash Memory(플래시메모리), File system(파일시스템), mount (마운트), recovery(복구)

1. 서 론

최근의 컴퓨터 산업은 유비쿼터스 컴퓨팅 환경을 지향하는 추세에 따라서 이동전화, 개인용 정보 단말기(PDA), MP3 재생기, PMP 등의 다양한 이동 가능한 정보기기들이 널리 보급되고 있다. 이에 따라서

※ 교신저자(Corresponding Author) : 이태훈, 주소 : 부산광역시 금정구 장전동 산30번지(609-735), 전화 : 051)518-7502, FAX : 051)517-2431, E-mail : withsoul@pusan.ac.kr
접수일 : 2007년 12월 27일, 완료일 : 2008년 4월 23일
[†] 삼성전자 정보통신총괄 통신연구소 A/V코덱랩
(E-mail : waller@pusan.ac.kr)
[††] 준회원, 부산대학교 컴퓨터공학과 박사과정
[†††] 준회원, 부산대학교 컴퓨터공학과 석사과정
(E-mail : withsoul@pusan.ac.kr, cat@pusan.ac.kr)
[††††] 종신회원, 부산대학교 컴퓨터공학과 교수
(E-mail : kdchung@pusan.ac.kr)

이러한 이동 정보기기에서 데이터를 저장하고 처리하기 위한 효율적인 저장장치가 필요하다. 일반적인 컴퓨터 시스템에서 사용되는 하드 디스크 기반의 저장장치는 부피가 크고, 외부 충격에 대해 내구성이 취약하며, 기계적인 동작이 필요하기 때문에 소비전력과 응답 시간이 크다는 단점이 존재한다. 따라서 물리적인 충격에 강해 휴대가 용이하고 집적도가 높아 소형화 기기에 적합하며, 전력을 적게 소모해 동일 배터리 전력으로 장시간 동작 가능한 특징을 가진 NAND 플래시 메모리가 데이터 저장 매체로서 사용하는 사례가 증가 하고 있다[1].

그러나 NAND 플래시 메모리[2]는 기존의 저장 매체인 디스크와 달리 하드웨어적인 제약이 존재한다. 첫째, 플래시 메모리는 데이터 수정 시 본래 주소에 덮어쓰기가 불가능하다. 둘째, 플래시 메모리의 각 블록은 초기화 연산의 횟수가 10만 ~ 100만 번으로 제한되어 있기 때문에 플래시 메모리의 전체 공간이 균등하게 사용되지 못하는 경우에 사용 가능한 메모리 가용 공간이 급격히 줄어들게 된다[3]. 셋째, 읽기와 쓰기 연산은 페이지(page) 단위로 처리되는 반면, 지움 연산은 일정한 개수의 페이지로 구성된 블록(block) 단위로 처리된다.

이러한 NAND 플래시 메모리의 특성으로 인하여 기존 하드디스크 파일 시스템을 바로 적용할 수 없으므로 NAND 플래시 메모리용 파일 시스템에 대한 연구가 활발히 진행 되어 왔다. 대표적인 NAND 플래시 메모리 파일 시스템으로 YAFFS(Yet Another Flash File system)[4]과 RFFS(Reliable Flash File System)[5,6]가 있다. YAFFS는 플래시 메모리 전체 영역을 읽어 부팅을 하게 되어 플래시 메모리 용량이 커지게 되면 부팅 시간이 느린 단점이 있다. 또한 비정상 종료와 같은 문제가 생겼을 경우 빠른 복구를 위한 방법이 필요하게 된다[7]. 이와 같은 문제점을 해결하기 위해 마운팅 시에 플래시 메모리의 크기에 관계없이 일정한 마운트 시간을 제공하기 위해 제안된 파일 시스템으로 RFFS가 있다. 빠른 마운트 및 복구를 제공하지만 이를 위해 많은 메타데이터를 사용하는 문제가 있다. 그리고 YAFFS와 RFFS의 쓰기 연산에 대한 새로운 블록을 할당할 때 전체 플래시 메모리 정보를 순차적으로 읽기에 연산 성능이 느려지는 문제점이 있다.

본 논문에서는 이와 같은 기존 파일 시스템의 문제점을 해결하기 위한 효율적이고 신뢰성 있는 플래시 파일 시스템(ERFFS : Efficient and Reliable Flash File System)을 제안한다. 제안하는 NAND 플래시 메모리용 파일 시스템은 최소한의 메타데이터를 이용한 빠른 마운트와 복구를 지원하는 구조와 불연속적인 블록에 대한 할당 시간을 줄이는 할당기법을 제안하고 있다. 실험에서는 ERFFS의 마운팅과 복구 시간 그리고 할당성능을 YAFFS와 RFFS의 성능과 비교하여 우수성을 검증하였다.

본 논문의 구성은 다음과 같다. 2 장에서는 플래시 메모리와 대표적인 NAND 플래시 파일 시스템인 YAFFS와 RFFS에 대해 기술한다. 3 장에서는 본 논문에서 제안하는 ERFFS에 대해 설명하며, 4 장에서는 ERFFS의 성능을 실험을 통해 평가하였다. 5장에서 이 논문의 결론과 향후 과제를 제시한다.

2. 관련연구

2.1 플래시 메모리

플래시 메모리는 일종의 EEPROM(Electrically Erasable and Programmable ROM)으로 비휘발성의 메모리 반도체로 관련 기술도 활발히 연구되고 있다[8].

표 1은 저장 매체별 각 연산의 처리시간을 나타내고 있다[9]. 플래시 메모리는 기존의 저장 매체인 디스크에 비해 매우 빠른 연산 속도를 가지고 있는 것을 알 수 있다. 그리고 플래시 메모리는 다른 저장 매체와 달리 지움 시간(Erase)이 존재하는 것을 확인할 수 있는데 지움 연산 시간이 읽기나 쓰기 연산 시간에 비해 많이 소모 되는 것을 확인 할 수 있다.

NOR 플래시 메모리는 바이트 단위접근이 가능하며 집적도가 낮고 고가이기 때문에 소스코드 저장용으로는 사용되며, NAND 플래시 메모리는 대용량이

표 1. 저장 매체별 처리속도

저장매체	처리시간(512Bytes)		
	Read	Write	Erase
DRAM	2.56us	2.56us	N/A
NOR flash	14.4us	3.53ms	1.2s(128KB)
NAND flash	35.9us	226us	2ms(16KB)
DISK	12.4ms	12.4ms	N/A

며 비용이 저렴하기 때문에 데이터 저장용으로 사용하고 있다. NAND 플래시 메모리는 한 메모리 셀 당 2비트를 나타낼 수 있는 MLC(Multi-Level Cell) NAND 플래시 메모리의 경우 블록 크기가 512KB, 페이지 크기가 4KB인 제품도 있다[10].

하지만 NAND 플래시 메모리는 다음과 같은 하드웨어적인 제약이 있다. 첫째, 플래시 메모리는 제자리 덮어쓰기(in-place-update)가 불가능하다. 둘째, 읽기와 쓰기 연산은 페이지 단위로 처리되는 반면, 지움 연산은 일정한 개수의 페이지로 구성된 블록 단위로 처리된다. 따라서 특정 페이지의 갱신된 내용을 기존 위치에 저장하기 위해서는 해당 블록에 지움 연산을 수행한 후, 같은 블록에 속한 페이지들의 내용도 다시 기록해야 한다. 셋째, 플래시 메모리의 각 블록은 지움 연산의 횟수가 제한되어 있다. 한 블록의 최대 지움 가능 횟수는 10만 ~ 100만 번이다. 따라서 특정 블록들에 지움 연산이 집중되어 플래시 메모리의 전체 공간이 균등하게 사용되지 못하면 플래시 메모리의 수명이 단축된다.

이러한 플래시 메모리의 특성을 이해하고 본 논문에서는 NAND 플래시 메모리를 대용량 멀티미디어 데이터 저장 매체로 사용하기 위한 NAND 플래시 메모리 전용 파일 시스템을 제안한다.

2.2 NAND 플래시 메모리 전용 파일 시스템

2.2.1 YAFFS

NAND 플래시 메모리를 위한 대표적인 파일 시스템으로는 YAFFS가 있다. YAFFS는 2002년 Aleph one 사에서 개발된 NAND 플래시 전용 파일 시스템이다.

YAFFS는 플래시 메모리의 제자리 덮어쓰기가 되지 않는 문제점을 해결하기 위하여 로그 구조 파일 시스템(LFS : Log-Structured File System)[11]을 기반으로 하였다. 따라서 파일에 대한 갱신 연산을 추가 쓰기(append) 연산으로 변형하여 처리하는 외부 갱신 기법(out-place-update)을 사용한다[12]. 갱신 연산이 빈번하게 발생할 경우, 한 파일의 데이터가 저장된 페이지들이 플래시 메모리 상에 흩어지게 된다. 따라서 마운팅 시에 전체 플래시 메모리를 읽어서 파일 시스템을 마운트한다.

그림 1는 플래시 메모리 상에서의 YAFFS 구조를

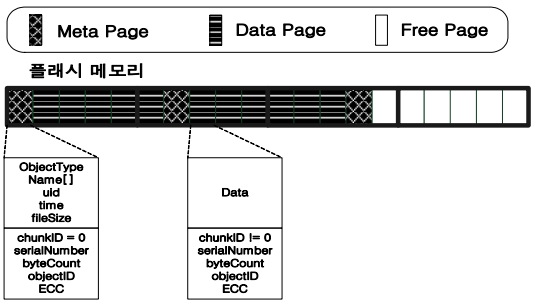


그림 1. YAFFS의 플래시 메모리상의 구조

보여준다. 1개 블록은 5개의 페이지로 구성된다고 가정한다. YAFFS는 페이지의 데이터 영역을 청크(chunk)라고 부르며, 청크에 실제 데이터를 저장한다.

마운팅 시에 스페어 영역의 정보를 읽고 메타 페이지인 경우 해당 파일의 Inode를 생성하고, 데이터 페이지인 경우 해당 파일의 Inode를 찾아서 해당 데이터 위치를 추가하게 된다. 이러한 과정을 전체 플래시 메모리에서 수행하여 파일 시스템의 마운트를 수행한다. 그리고 메인 메모리(RAM)상에 전체 블록의 상태를 관리하는 BlockInfo 구조체를 유지하며 할당 및 가비지 콜렉션(Garbage Collection)을 수행한다.

따라서 YAFFS는 다음과 같은 단점들이 있다. 첫째, 전체 플래시 메모리를 읽어 파일 시스템의 마운트를 수행하기에 많은 시간이 걸리게 된다. 이는 플래시 메모리 용량이 커질수록 증가하게 된다. 둘째, 빈 블록을 할당할 때 BlockInfo 구조체를 순차적으로 검사하므로 플래시 메모리 사용이 많아지면 빈 블록을 검색하는 시간이 많이 걸리게 된다.

2.2.2 RFFS

RFFS는 빠른 마운팅의 지원을 목적으로 하면서 효율적인 균등 사용 정책을 적용하고 저널링을 지원하도록 파일 시스템의 설계를 하였다.

그림 2는 플래시 메모리상에 RFFS를 구조를 나타내고 있다. RFFS는 크게 위치 정보 영역과 데이터 영역으로 구성된다.

RFFS의 위치 정보 영역은 정상 종료 여부를 나타내는 'MountFlag', 저널 로그 블록의 위치를 나타내는 'journal log', 블록 정보 블록의 위치를 나타내는 'block_info', 메타데이터 블록의 위치를 나타내는 'meta_data'로 구성이 된다. 파일의 정보를 가지고 있는 메타 데이터 블록을 통해 마운트 과정에서 파일

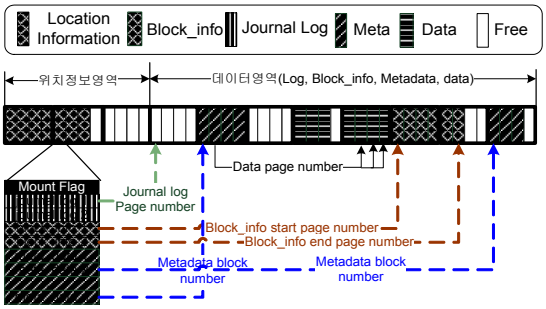


그림 2. RFFS 파일 시스템 구조

리스트를 구축하게 된다. 메타 데이터 블록에 저장되는 메타 정보는 YAFFS의 메타 정보와 달리 해당 파일의 모든 데이터 페이지의 위치 정보를 저장하고 있다. 따라서 플래시 메모리의 전체 블록을 읽지 않고 메타 데이터 블록만을 읽어서 마운트를 수행할 수 있다. 그러나 RFFS는 다음과 같은 단점들이 존재한다. 첫째, 메타 정보가 해당 파일의 모든 데이터 페이지의 위치를 기록하기 때문에 부분적인 갱신에도 메타 페이지의 갱신 작업이 필요하므로 연산 성능 저하가 발생한다. 둘째, 블록 정보 블록에 기록된 메타 데이터 블록과 위치 정보 영역에 기록된 메타데이터 블록의 정보가 중복되기에 플래시 메모리를 비효율적으로 사용하고 있다. 셋째, 블록 정보 영역에 매번 전체 플래시 메모리의 블록 정보를 기록하게 됨으로서 플래시 메모리 사용량이 많아진다.

3. ERFFS의 설계

3.1 파일 시스템 구조

그림 3은 ERFFS의 파일 시스템 구조를 나타내고 있다. 전체 플래시 메모리의 물리적인 공간을 두 부분으로 구분한다. 플래시 전체 블록의 정보를 저장하고 있는 플래시 정보영역(Flash Info Area)과 메타데이터와 데이터를 저장하는 플래시 데이터 영역(Flash Data Area)으로 구분된다.

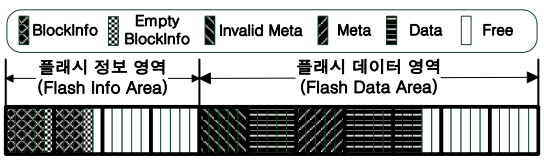


그림 3. ERFFS 파일 시스템 구조

3.1.1 플래시 정보 영역

플래시 정보 영역에 유지되는 정보는 플래시 메모리의 블록 정보로 구성이 된다. 블록 정보 객체는 블록 번호, 블록의 지움 횟수, 블록에 지워지지 않은 불필요한 페이지의 수, 블록에서 사용 중인 페이지의 수, 블록의 종류 그리고 블록의 상태 정보로 구성 된다. 그리고 언마운트 시에 메모리(RAM) 상에 유지하고 있는 Block_Info에 저장되어 있는 블록 정보 중 빈 블록 정보(emptyBlockInfo)를 모아서 플래시 정보 영역에 기록한다. 빈 블록 정보는 블록 번호를 나타내는 'block_num', 해당 블록의 지움 연산 횟수를 나타내는 'EraseCount' 그리고 연속된 블록을 표현하기 위한 'continuity'로 구성된다. 그림 4는 ERFFS의 플래시 정보 영역을 보여 준다.

3.2.1 플래시 데이터 영역

플래시 데이터영역은 플래시 정보영역을 제외한 나머지 영역으로 구성되고 이 영역은 메타데이터 및 데이터를 저장하는데 사용된다. 블록은 메타데이터, 무효메타데이터, 데이터 및 빈 블록으로 구분 된다. 메타데이터 블록은 파일 정보를 담고 있다. RFFS의 메타 데이터와 같이 ERFFS는 메타 데이터에도 파일의 모든 데이터 페이지의 위치를 저장한다. 무효메타데이터 블록은 무효화가 될 메타데이터를 저장하는 블록이다. 그리고 실제 데이터를 저장하는 데이터 블록과 초기화되어 있는 빈 블록으로 구성된다.

3.2 메모리상의 구조

플래시 메모리 파일 시스템이 동작하기 위해서 메모리(RAM)상에 정보를 유지 하게 된다. 그림 5은 ERFFS의 동작을 위한 메모리상에 유지되는 구조체

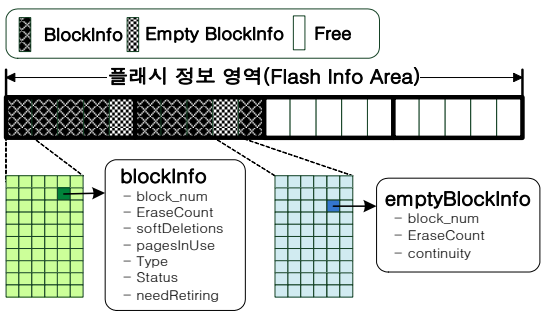


그림 4. 플래시 정보 영역의 구조

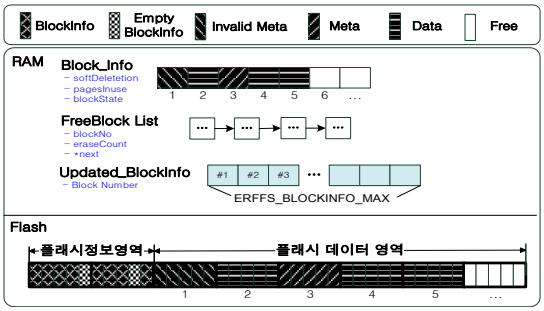


그림 5. ERFFS의 메모리상의 구조

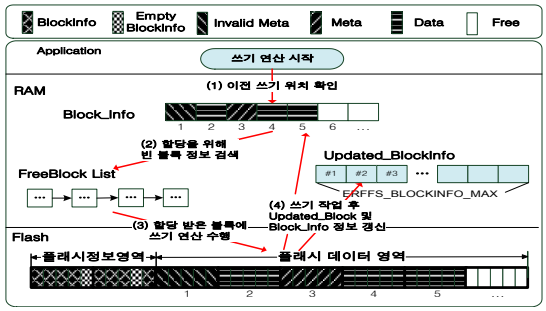


그림 6. ERFFS의 쓰기 연산 과정

를 보여준다. 전체 블록을 관리하기 위한 Block_Info 정보가 먼저 구성된다. Block_Info 정보는 전체 블록에 대하여 각 블록에 무효페이지의 수를 나타내는 'softDeletion', 해당 블록에서 사용 중인 페이지의 수를 나타내는 'pagesInUse' 그리고 해당 블록의 상태를 나타내는 'blockState'로 구성 된다.

이전 언마운트 과정에서 기록했던 빈 블록 정보를 이용하여 마운트 과정에서 빈 블록 연결 리스트(FreeBlock List) 형태로 구성하게 된다. 빈 블록 연결 리스트의 각 노드는 빈 블록의 위치, 블록 지움 연산 횟수와 다음 노드를 가리키는 포인터로 구성된다. 빈 블록 연결 리스트는 블록 지움 연산 횟수를 이용하여 블록 지움 연산 횟수별 연결 리스트를 유지한다. 이 정보를 이용하여 빈 블록 할당 요청의 경우 블록 지움 연산이 작은 연결 리스트의 빈 블록을 먼저 할당하게 함으로서 불연속적으로 존재하는 빈 블록에 대한 빠른 할당을 지원한다. 연결 리스트를 위한 추가적인 메모리 사용이 필요하게 되지만 플래시 메모리 용량이 커지고 사용량이 많아질수록 기존 YAFFS의 순차적 검색 할당 방법에 비해 빠른 할당이 가능하다. 플래시 메모리는 기존 하드디스크 파일 시스템과 달리 위치에 상관없는 빠른 입출력을 할 수 있다는 점을 착안 할 때 연결 리스트를 이용한 구조로 인해 하드디스크 파일 시스템에서 고려해야 했던 파일 위치에 따른 헤더 이동에 따른 요소는 고려하지 않아도 되므로 효율적이라 할 수 있다.

3.3 ERFFS 연산 과정

그림 6은 ERFFS의 쓰기 연산 과정을 보여 준다. 이전에 마지막으로 쓰인 블록의 위치를 확인하고 새로운 블록의 할당이 필요하게 되면 FreeBlock List로부터 새로운 블록을 할당 받게 된다. 할당 받은 블록을 통해 쓰기 과정을 수행하고 쓰기 작업이 완료된 후에 Updated_BlocInfo 및 Block_Info 정보를 갱신한다.

그림 7과 같이 ERFFS은 플래시 메모리의 블록 중 사용된 블록의 정보를 부분적으로 기록하는 방법을 사용한다. 그림 4와 같이 ERFFS_BLOCKINFO_MAX 개의 blockInfo 구조체가 하나의 페이지를 구성하게 된다. 메모리상의 Updated_BlockInfo 리스트가 한 페이지에 들어가는 블록의 개수인 ERFFS_BLOCKINFO_MAX 개가 되면 하나의 페이지로 쓰기 작업을 수행한다. 갱신된 블록의 정보는 block-Info 구조체로 유지 하는 것이 아니라 블록 번호만을 유지하고 쓰기 시점에 해당 Block_Info 정보만을 읽어 쓰기 작업을 하게 된다. RFFS의 경우 블록 정보를 관리할 때 플래시 전체 영역에 대한 블록 정보를 기록하게 된다. 그러나 ERFFS는 앞서 설명한 것과 같이 갱신된 블록 정보를 기록함으로서 플래시 메모리 공간을 절약할 수 있고 마운팅 시간도 줄일 수 있게 된다.

그림 4의 emptyBlockInfo는 빈 블록에 관해서 블

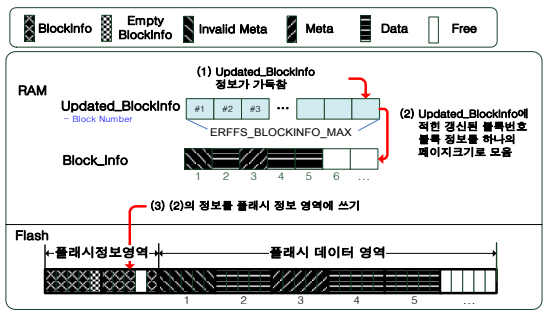
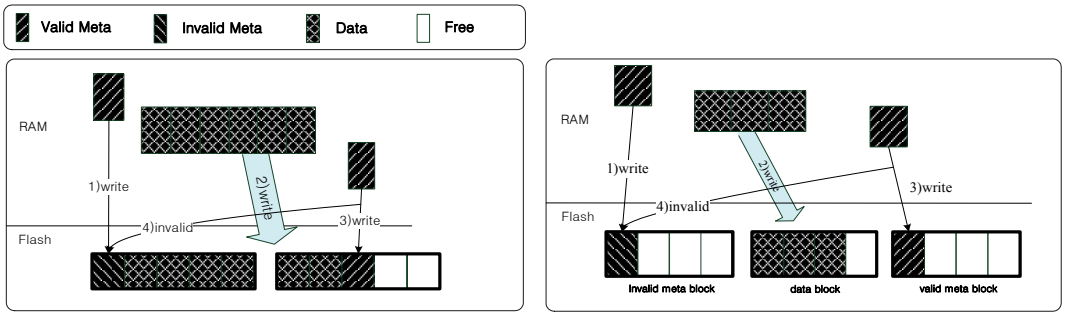


그림 7. 플래시 정보 영역 쓰기 과정



(a) YAFFS에서의 쓰기 작업 (b) 제안하는 쓰기 작업

그림 8. 파일 쓰기 작업

록 번호와 해당 블록의 지움 횟수에 관한 데이터를 기록한 것이다. 빈 블록을 관리할 때에 최소한의 공간에 최대한의 빈 블록 정보를 유지하기 위해 연속된 빈 블록의 경우 'continuity'를 사용하여 플래시 메모리 사용을 최소화하였다.

플래시 데이터 영역은 메타데이터 블록, 무효메타데이터 블록, 데이터 블록과 빈 블록으로 구성되어 있다. 디렉터리, 하드 링크와 심볼릭 링크 생성 시, 메타데이터가 메타데이터 블록에 저장된다. 파일 생성 시에는 데이터가 데이터블록에 저장되고 데이터 위치를 포함한 메타데이터가 메타데이터 블록에 저장된다.

그림 8은 파일 쓰기 작업을 하는 과정을 나타내고 있다. 그림 8 (a)에서 보는 것과 같이 YAFFS의 경우 무효화 될 메타 페이지와 그렇지 않은 메타 페이지의 구분이 없이 같은 블록에 쓰이게 된다. 그래서 무효화 될 메타 페이지는 여러 블록에 퍼져 존재하게 된다. 이것은 블록 지움 연산 시에 무효화 페이지를 증가시켜 시스템 성능을 저하 시키고 마운팅시 불필요한 메타 정보를 읽게 하여 마운팅 시간을 지연시킨다. 그림 8 (b)에서 보는 것과 같이 파일 쓰기 작업 시에 무효화될 메타데이터를 따로 관리한다. 무효메타데이터 영역은 무효화된 메타 데이터만 존재하므로 무효메타데이터 블록이 가득 차게 되면 바로 블록 지움 연산을 수행하여 가비지 컬렉션 작업에 오버헤드를 줄일 수 있다. 그리고 마운팅시에도 불필요한 메타데이터 정보를 읽지 않게 하여 마운팅 성능에도 향상시킨다.

3.4 빠른 마운팅 기법

그림 9는 ERFFS의 마운팅 과정을 보이고 있다.

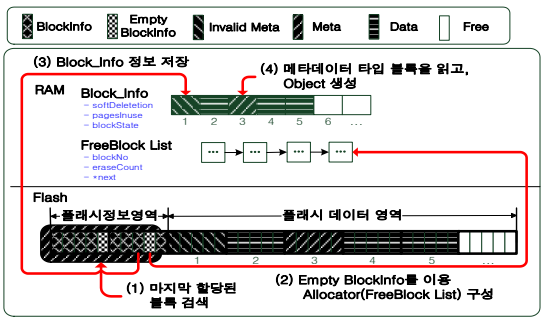


그림 9. ERFFS의 마운팅 과정

ERFFS는 빠른 마운팅을 지원하기 위하여 플래시 정보 영역에 사용 중인 블록 정보를 저장하고 있다. 저장한 플래시 블록 정보를 마운팅 과정에서 읽고 메타데이터 블록을 빠르게 찾음으로써 빠른 마운팅을 지원할 수 있다.

ERFFS는 RFFS와 같이 메타데이터에 데이터 위치를 가지고 있어 데이터를 읽지 않고 메타데이터만을 읽어 빠른 마운팅이 가능하다. RFFS에서는 전체 무효메타데이터와 메타데이터가 섞여있는 메타데이터 블록을 다 읽어야하지만 ERFFS는 무효메타데이터 블록을 사용함으로서 읽어야 할 메타데이터 블록을 최소화하였다.

3.5 빠른 복구 기법

그림 10은 ERFFS의 복구 과정을 보이고 있다. ERFFS는 마운트 시에 블록 정보를 플래시 정보 영역만 읽어 마운트를 수행하기 때문에 플래시 정보 영역에 블록 정보가 완전히 기록되지 않은 상태로 비정상 종료 하게 되면 마운트 시에 읽은 블록 정보와 실제 블록 정보가 불일치하게 된다. 이를 해결하

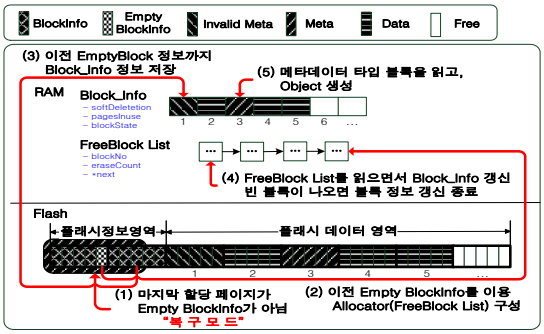


그림 10. ERFFS의 복구 과정

기 위해 비정상 종료 후 마운트 시에 전체 플래시 메모리를 읽는 방법이 있으나 그렇게 되면 비정상 종료 시에는 마운트 시간이 느려진다. 이를 해결하기 위해 언마운트 시에 기록하는 빈 블록 정보(EmptyBlock Info)를 이용한다. ERFFS는 마운트 시에 빈 블록 정보를 확인 하는 것으로 파일 시스템의 비정상 종료 여부를 확인하게 된다. 즉, 빈 블록 정보를 로그로 활용하게 된다. 언마운팅시에 항상 빈 블록 정보를 적게 되는데 마운팅시에 마지막에 할당된 블록의 마지막 페이지가 빈 블록 정보가 아니면 복구 모드를 수행하게 된다.

복구 모드는 이전에 기록했던 빈 블록 정보를 이용하여 FreeBlock List를 구축하고, 이를 이용하여 비정상 종료로 잃어버린 블록 정보를 복구한다. 빈 블록 정보는 마운트 시에 할당자를 구성하는데 활용되며 이전 정보로 할당자를 구성하게 되면 실제 사용한 빈 블록 정보와 불일치하게 되므로 할당자를 이용하여 실제 블록 정보와 일치 할 때 까지 읽어 블록 정보를 복구한다. 할당자는 블록의 지움 연산 횟수를 우선순위로 구성되기에 같은 빈블록 정보를 가지면 항상 같은 순서로 구성이 되게 된다. 이때 같은 지움 연산 횟수의 경우는 순차적으로 구성되므로 순서는 항상 같게 된다.

4. 실험 환경 및 성능 평가

본 장에서는 ERFFS의 성능 평가를 위한 실험 환경과 그 결과를 살펴보도록 한다. ERFFS의 마운팅과 복구, 할당 성능 그리고 파일 연산에 소요 된 시간을 관련 연구에서 살펴본 YAFFS, RFFS와 비교하겠다.

4.1 실험 환경

본 논문에서 제안한 ERFFS의 성능 평가를 위하여 ERFFS를 구현하고 실험을 수행하였다. 실험은 휴인스의 PXA255-Pro III 임베디드 보드를 사용 하였다[13]. 실험보드의 사양은 표 2와 같다. 64MB NAND 플래시 메모리에서 성능평가 실험을 하였다. 플래시 메모리는 소블록 NAND 플래시 메모리로 크기를 10MB로 분할하여 사용하였고 운영체제로는 리눅스 2.4.19를 사용하였다.

4.2 마운팅 성능

플래시 메모리 파일 시스템의 마운팅 시간은 플래시 메모리의 사용률에 따라 영향을 받는다. 따라서 마운팅 성능의 평가를 위해 플래시 메모리의 사용률을 0%에서 90%까지 10%씩 증가시키면서 마운팅 시간을 측정하였다. 10Kbytes 크기의 파일을 생성하면서 사용량을 증가시켰다.

그림 11은 플래시 메모리 사용률에 따른 YAFFS, RFFS, ERFFS의 마운팅 시간을 보여 주고 있다. YAFFS의 경우, 마운팅 과정에서 전체 플래시 메모리를 읽기 때문에 플래시 메모리의 사용량에 비례하여 마운팅 시간이 증가하고 있다. RFFS는 무효메타데이터의 구분 없이 모든 메타데이터 정보를 읽게

표 2. PXA255-Pro III 임베디드 보드

항 목		명 세
CPU		Intel PXA255(Xscale)
SDRAM		128 MB SDRAM
Flash Memory	NOR	32 MB (Intel)
	NAND	64 MB (Samsung)

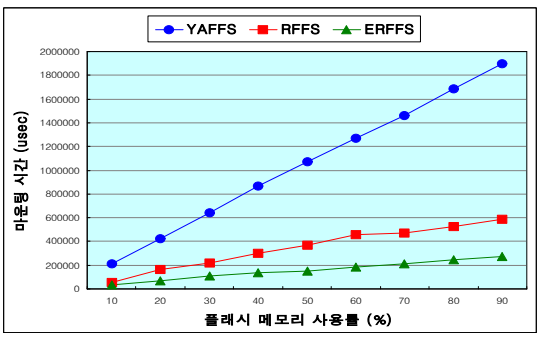


그림 11. 플래시 메모리 사용량에 따른 마운트 시간

되고 저널링 로그 정보 블록 및 전체 블록 정보를 읽기에 YAFFS에 비해 성능은 많이 향상 되었으나 ERFFS에 비교해 성능이 느린 것을 확인 할 수 있다. ERFFS는 플래시 정보 영역을 읽고 메타데이터 블록만을 읽는 과정으로 마운팅을 수행하게 되고 마운팅 시간은 메타데이터 블록의 수에 비례하여 마운팅 시간이 증가하게 된다. ERFFS는 마운팅 시간이 YAFFS에 비해 약 83% ~ 86% 감소하였고 RFFS에 비해 약 38% ~ 60% 감소 시켰다.

4.3 복구 성능

ERFFS는 비정상 종료에 대한 여부를 판단하고 비정상 종료로 확인되면 복구모드로 동작하게 된다. 그림 12는 복구 보드가 동작한 경우 마운트 시간을 측정한 것이다. 정상 종료에 비해 갱신되지 않은 블록 정보를 갱신하기 위한 시간이 추가로 소모 되고 있지만 RFFS에 비해 약 20%~50%의 시간으로 비정상 종료시에도 빠르게 복구 모드로 마운팅 한 것을 알 수 있다.

4.4 할당 성능

ERFFS는 마운팅시에 할당을 위한 빈 블록 정보를 이용하여 FreeBlock_List를 구축한다. 기존에 YAFFS나 RFFS의 경우 빈 블록 할당을 위해서 플래시 메모리를 라운드 로빈(round-robin) 방식으로 플래시 메모리 블록 정보를 하나씩 읽어 가면서 빈 블록을 확인하게 된다. 이러한 순차적인 블록 접근 방식은 플래시 메모리의 사용량이 높아 빈 블록이 적고 플래시 메모리 상에 많이 흩어져 있는 경우, 빈 블록 할당 시에 빈 블록 검색에 많은 시간을 소요

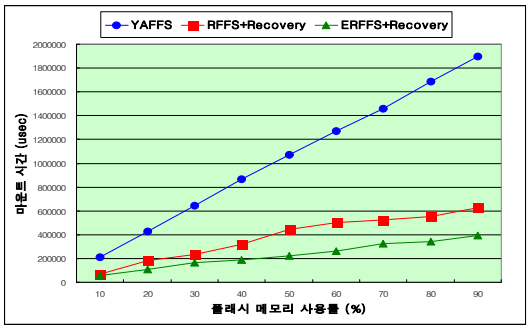


그림 12. 플래시 메모리 사용량에 따른 복구 시간

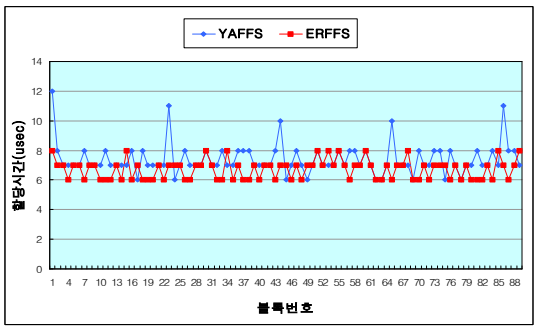


그림 13. 빈블록 할당 성능

하게 된다. 할당 성능 측정은 플래시 메모리의 빈 블록 분포에 따른 영향을 받으므로 임의로 빈 블록의 분포를 조정하여 실험을 하였다.

그림 13은 플래시 메모리의 빈 블록을 20개 블록 단위로 설정하였다. 기존의 YAFFS의 경우 빈 블록이 연속적이지 않은 경우에 할당 시간이 증가 하는 것을 확인 할 수 있다. 20개 블록 단위로 빈 블록을 성정한 경우 기존 방법에 비해 10%의 할당 성능 향상을 보였다. 플래시 메모리의 용량이 커지고 사용량이 많아질수록 빈 블록 할당을 위한 블록 간의 거리에 대한 검색 오버헤드가 커지게 되므로 더 많은 시간을 소요하게 된다. 그러므로 ERFFS는 플래시 메모리의 크기에 비례하여 성능 향상을 기대 할 수 있다.

4.5 파일 연산 성능

ERFFS는 파일 연산을 위해서 파일 동작 시에 플래시 정보 영역에 블록 정보를 유지하기위해서 BlockInfo 페이지를 쓰는 작업을 수행하게 된다. 파일 시스템을 유지하기 위해 동작하게 되는 연산이 많이 일어나게 되면 파일 시스템의 동작에 있어 성능 저하가 오게 된다. 이를 확인하기 위하여 파일이 적히게 될 때의 파일 시스템을 유지하기 위해서 추가적으로 적히게 되는 페이지 수를 YAFFS, RFFS와 비교 하여 파일 시스템 관리를 위한 추가 연산에 대하여 알아보겠다.

실험은 64MB 크기의 NAND 플래시 메모리에 10MB의 데이터가 존재하는 경우 데이터 정보를 비교한다. 이 때 데이터의 크기를 1KB, 10KB, 100KB, 1MB로 하여 실험 하였다. 즉 1KB 파일 10000개, 10KB 파일 1000개, 100KB 파일 100개 그리고 1MB 파일 10개를 생성 했을 때의 사용 페이지의 개수를

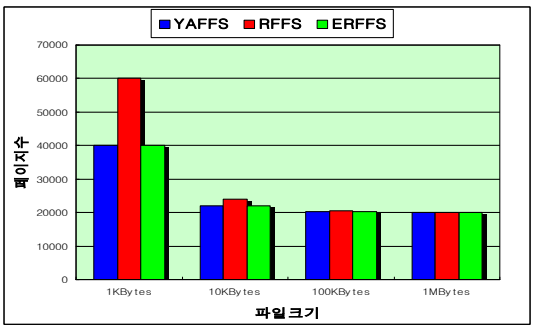


그림 14. 파일크기에 따른 페이지 생성 개수

나타내고 있다.

그림 14에서 보는 것과 같이 파일 크기에 따른 페이지 생성 개수를 확인해 본 결과 YAFFS와 비교해 볼 때 ERFFS는 파일 크기에 상관없이 추가 생성된 페이지가 거의 없는 반면 RFFS는 작은 파일에서의 추가적으로 생성 되는 페이지가 많아져서 성능 저하가 오게 된다. ERFFS는 빠른 마운팅 및 부팅을 위한 구조를 제안하고 있지만 파일시스템이 동작하는 과정에서 오버헤드는 거의 없는 것을 확인 할 수 있다.

5. 결 론

본 본문에서는 NAND 플래시 메모리를 위한 파일 시스템을 설계하였다. 본 논문에서 제안한 ERFFS는 기존의 YAFFS가 가지고 있는 느린 마운팅을 해결하고 RFFS 보다 향상 된 마운팅 성능을 가지는 구조를 제안하였다. 또한 불연속적인 블록에 대한 빠른 할당을 위한 할당 기법을 제안하였고 비정상 종료 시에도 빠른 복구가 가능하도록 설계하였다.

플래시 메모리 연산 과정에 갱신 된 블록 정보를 플래시 정보 플래시 정보 영역에 기록하고 언마운팅 시에 빈 블록 정보를 기록하는 방법을 통하여 플래시 정보 영역만을 읽어 마운팅이 가능 하도록 하였다. 실험 결과 YAFFS에 비해 마운팅 시간을 83% ~ 86% 감소 시켰고, RFFS에 비해 약 38% ~ 60% 감소 시켰다. 또한 플래시 정보 영역에 기록된 빈 블록 정보를 이용하여 빠른 복구를 지원하여 RFFS에 비해 약 20%~50%로 성능이 향상 되었다. 그리고 할당 성능도 기존의 순차적인 검색 할당 방법에 비하여 10%의 성능 향상을 보였고, 할당 시에 블록 지움 연산 횟수를 고려하여 균등 사용을 지원하였다.

본 논문은 지움 연산 횟수를 고려한 할당을 통해 균등 사용을 지원하고 블록의 타입을 구분함으로서 지움 연산을 효율적으로 지원하고 있으나 능동적인 가비지 컬렉션 방법이 향후 연구 되어야 한다.

참 고 문 헌

[1] F.Douglis, R. Caceres, F. Kaashoek, K. Li, B. Marsh, and J.A. Tauber, “Storage Alter-natives for Mobile Computers,” Proceedings of the 1st Symposium on Operating Systems Design and Implementation(OSDI), pp. 25-37, 1994.

[2] White Paper : Two Technologies Compared: NOR vs. NAND, “http://www.data-io.com/pdf/NAND/MSystems/MSystems_NOR_vs_NAND.pdf”.

[3] S.j. Baek, S.j. Ahn, J.M. Choi, D.H. Lee, S.H. Noh, “Uniformity Improving Page Allocation for Flash Memory File Systems,” EMSOFT07, pp. 154-163, 2007.

[4] Alepha One Company, “The Yet Another Flash File System(YAFFS),” http://www.yaffs.net/

[5] 박송화, 이태훈, 정기동, “A Flash File System to Support Fast Mounting for NAND Flash Memory Based Embedded Systems,” SAMOS 2006, Lecture Notes in Computer Science, Vol.4017, pp. 415-424, 2006.

[6] 박송화, 이주경, 정기동, “임베디드 기기를 위한 NAND 플래시 메모리 파일 시스템의 설계,” 정보과학회 2005 추계학술대회발표대회, pp. 151-153, 2005.

[7] K. S. Yim, J. H. Kim, and K. Koh, “A Fast Start-Up Technique for Flash Memory Based Computing Systems,” ACM Symposium on Applied Computing, pp. 852-858, March 2005.

[8] 김정기, 박승민, 김채규, “정보가전을위한 플래시 파일 시스템에서 등급별 지움 정책과 오류 복구 방법,” 제5회 차세대 통신 소프트웨어 학술대회(NCS1), pp. 938-941, 2001.

[9] J.S. Kim, J.M. Kim, S.H. Noh, S.L. Min, Y.K.

Cho, “A Space-efficient Flash Translation Layer for CompactFlash Systems,” IEEE Transactions on Consumer Electronics, Vol 48, pp. 366-375, 2002.

[10] “Implemeting MLC NAND Flash for Cost-Effective, High-Capacity Memory,” M-Systems Wite Paper, 2003, “http://www.data-io.com/pdf/NAND/MSystems/Implementin g_MLC_NAND_Flash.pdf”

[11] M. Resenblum and J. K. Ousterhout, “The Design and Implementation of a Log-Structured File System,” ACM Transaction on Computer Systems, Vol.10, pp. 26-52, July 1992

[12] 진종원, 이태훈, 정기동, “NAND 플래시 메모리 파일 시스템의 빠른 연산을 위한 설계 및 성능 평가,” 한국정보과학회 2007 한국컴퓨터종합학술대회, pp. 273-274, 2007.

[13] 휴인스 기술연구소, Intel PXA255와 임베디드 리눅스 응용 [제2판], 홍릉과학출판사, 2004.

진 종 원

2006년 2월 부산대학교 정보컴퓨터공학과 학사
2008년 2월 부산대학교 컴퓨터공학과 석사
2008년 2월~ 현재 삼성전자 정보통신총괄 통신연구소 A/V코덱랩
관심분야 : 내장형 시스템, 파일 시스템, 코덱

이 태 훈

2004년 2월 부산대학교 정보컴퓨터공학과 학사
2006년 2월 부산대학교 컴퓨터공학과 석사
2006년 2월~현재 부산대학교 컴퓨터공학과박사과정
관심분야 : 내장형 시스템, 파일 시스템

이 승 환

2007년 2월 부산대학교 정보컴퓨터공학과 학사
2007년 2월~현재 부산대학교 컴퓨터공학과 석사과정
관심분야 : 내장형 시스템, 파일 시스템

정 기 동

1973년 서울대학교 졸업(학사)
1975년 서울대학교 대학원 졸업(석사)
1986년 서울대학교 대학원 계산통계학과 졸업(이학박사)
1978년~현재 부산대학교 컴퓨터공학과 교수
관심분야 : 멀티미디어 시스템, 멀티미디어 통신, 병렬처리