■ 2007년도 학생논문 경진대회 수상작

NAND 플래시 메모리 기반 파일 시스템을 위한 빠른 마운트 및 안정성 기법

(A Fast Mount and Stability Scheme for a NAND Flash Memory-based File System)

박 상 오 † 김 성 조 ††
(Sang Oh Park) (Sung Jo Kim)

요 약 기존 NAND 플래시 메모리 기반 파일 시스템은 NAND 플래시 메모리 특성의 하나인 마모도로 인하여 삭제 횟수를 고려해야 하므로 특정 영역을 지속적으로 필요로 하는 정보는 파일 시스템에 저장될 수 없다. 이로 인하여 대부분의 NAND 플래시 메모리 파일 시스템은 마운트될 때 플래시 메모리 전체를 스캔하여 파일 시스템 구조를 파악한다. 따라서 마운트 시간은 NAND 플래시 메모리 크기에 따라 선형적으로 증가할 뿐만 아니라 NAND 플래시 메모리의 사용 형태에 따라 매우 달라 질 수 있다. 또한, NAND 플래시 메모리를 저장 장치로 많이 사용하는 모바일 기기는 특성상 안정적인 전원 공급을 보장받지 못하기 때문에 NAND 플래시 메모리 파일 시스템의 안정성 확보를 위한 대책이 요구 된다. 본 논문에서는 NAND 플래시 메모리 전용 파일 시스템의 마운트 시간을 향상시키고, 예기치 않은 정전과 같은 상황이 발생하여도 파일 시스템이 오동작하지 않고, 일관성 있게 복구가 가능하도록 설계하고 구현하였다. 구현된 파일 시스템은 기존의 NAND 플래시 메모리 기반 파일 시스템인 JFFS2에 비해 최대 19배, YAFFS와는 2배 정도 향상된 마운트 성능을 보였으며, 안정성 측면에서도 좋은 안정성을 가진 JFFS2와 같은 성능을 보였다.

키워드 : 리눅스, 파일시스템, NAND, 플래시 메모리, 마운트, 안정성

Abstract NAND flash memory-based file systems cannot store their system-related information in the file system due to wear-leveling of NAND flash memory. This forces NAND flash memory-based file systems to scan the whole flash memory during their mounts. The mount time usually increases linearly according to the size of and the usage pattern of the flash memory. NAND flash memory has been widely used as the storage medium of mobile devices. Due to the fact that mobile devices have unstable power supply, the file system for NAND flash memory requires stable recovery mechanism from power failure. In this paper, we present design and implementation of a new NAND flash memory-based file system that provides fast mount and enhanced stability. Our file system mounts 19 times faster than JFFS2's and 2 times faster than YAFFS's. The stability of our file system is also shown to be equivalent to that of JFFS2.

Key words : Linux, File system, NAND, Flash memory, Mount, Stability

· 본 연구는 중소기업청의 2006년 산·학·연 공동기술개발 컨소시엄사업의 연구결과로 수행되었음

† 학생회원 : 중앙대학교 컴퓨터공학부
sj1st@konan.cse.cau.ac.kr
†† 종신회원 : 중앙대학교 컴퓨터공학부 교수
sjkim@cau.ac.kr

논문접수 : 2007년 5월 29일
심사완료 : 2007년 11월 21일

: 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.

정보과학회논문지: 시스템 및 이론 제34권 제12호(2007.12)

Copyright@2007 한국정보과학회

1. 서 론

최근 주로 사용되는 플래시 메모리에는 NOR형과 NAND형 등 두 가지가 있다. 기존에 많이 사용되던 NOR 플래시 메모리는 집적도가 낮고 고가이며 읽기 속도가 빠르지만 쓰기 속도가 느리기 때문에 운영체제나 응용프로그램 등의 코드 저장에 많이 사용되고 있다[1]. NAND 플래시 메모리는 단일 칩으로도 대용량이고 NOR 플래시 메모리에 비해 비용도 저렴하며 쓰기 속도가 빠르기 때문에 멀티미디어 데이타 저장에 많이 사용되고 있다[2]. 따라서 NAND 플래시 메모리를 내장형 기기의 기본 저장장치로 사용하게 되면 기본 저장용량을 크게 할 수 있으며 기기의 비용도 낮출 수 있다. 또한, 최근 출시되는 많은 수의 마이크로 컨트롤러들은 NAND 플래시 메모리를 제어할 수 있는 기능이 탑재되어 있어 NAND 플래시 메모리로부터 직접 부팅이 가능하므로, 기기들이 NOR 플래시 메모리와 같은 부가적인 부트용 저장장치 없이 NAND 플래시 메모리만으로 저장장치를 구성할 수 있다. 하지만, 플래시 메모리는 디스크와 같은 기존 저장매체와 다른 몇 가지 제약 사항들이 있다. 첫 번째, 데이타가 기록된 영역에 새로운 데이타를 다시 쓰려면 먼저 그 영역을 지워야 한다. 두 번째, 삭제 연산 단위가 쓰기 연산 단위보다 크기 때문에 덮어 쓰기가 용이하지 않다. 마지막으로 삭제 연산의 횟수가 제한되어 있기 때문에, 특정 블록에 반복적으로 쓰기 연산이 수행된다면 전체 플래시 메모리의 수명과 성능을 떨어뜨리게 된다.

FFS(Flash File System)와 JFFS2(Journaling Flash File System 2)[3]와 같이 이러한 문제점을 개선한 플래시 메모리 전용 파일 시스템이 개발되었지만, NOR 플래시 메모리에 기반을 두어 NAND 플래시 메모리의 특성을 고려하지 못하였다. 최근에서야 JFFS2가 NAND 플래시 메모리 특성을 고려하여 수정되었고, YAFFS (Yet Another Flash File System)[4]와 같은 NAND 플래시 메모리 전용 파일 시스템이 개발되었다.

기존의 NAND 플래시 메모리 전용 파일 시스템들은 마운트 시간이 오래 걸린다. 이것은 NAND 플래시 메모리의 특성 중에 하나인 마모도를 고려하여 특정 블록이 집중적으로 사용되지 않도록 함으로써 사용 가능한 NAND 플래시 메모리의 용량이 줄어들지 않게 파일시스템이 설계되었기 때문이다. 이러한 방법 중 하나는 수시로 변하는 파일 시스템의 정보 등을 저장하지 않는 것이다. 이로 인하여 대부분의 NAND 플래시 메모리 파일 시스템은 마운트될 때 NAND 플래시 메모리 전체를 스캔하여 파일 시스템 구조를 파악한다. 따라서 이러한 기법의 마운트 시간은 NAND 플래시 메모리 크기에 따라 선형적으로 증가하며, NAND 플래시 메모리의 사용 패턴에 따라 매우 다를 수 있다. 또한, 마운트가 시스템의 부트 과정에서 이루어지는 경우에 시스템 부팅 시간을 크게 증가시키는 문제를 야기한다. 또한, 파일 시스템은 고장에 대비한 복구(Crash Recovery)를 고려해야 한다. 파일 시스템이 고장 나지 않도록 최대한 대비를 해야 하고, 예기치 않은 시점에 뜻하지 않게 고장이 발생한 경우에도 가능한 빠른 시간 안에 일관성 있게 복구되어야 한다[5].

본 논문에서는 기존 NAND 플래시 메모리 파일 시스템인 YAFFS를 기반으로 NAND 플래시 메모리의 특성을 고려하여 설계된 파일 시스템인 NFFiS(Nand Flash memory File system)에서 빠른 마운트와 안정성 보장 기법에 대해서 기술한다.

본 연구에서 제안한 기법은 마운트 시 스캔되는 영역을 최소화하기 위해 연속된 몇 개의 물리적인 블록을 하나로 묶어 논리적인 확장 블록으로서 ExBlock (Extended Block)이라는 새로운 블록 개념과 ExBlock에 존재하는 모든 페이지를 관리하는 테이블인 SNode (Snapshot Node)와 SNode 생성을 요청하는 VNode (Victim Node)를 제안하며, 마운트 시 SNode 만을 스캔 함으로써 마운트 시간을 획기적으로 줄일 수 있도록 한다. 또한, 갑작스런 전원 차단에 의해 파일 시스템이 안정적으로 종료되지 않았을 때 데이타 손실에 의한 오류가 발생할 수 있는데, 이를 위해 Fragment Version 개념을 도입하여 파일 시스템의 안정성을 향상시켰다.

본 논문의 구성은 다음과 같다. 2장에서는 기존 NAND 플래시 메모리 전용 파일 시스템의 문제점에 대해 알아보고, 3장에서는 제안한 마운트 기법과 안정성 기법에 대해서 설명한다. 4장에서는 실험을 통해 본 논문에서 구현한 마운트와 안정성 기법의 성능을 기존의 NAND 플래시 메모리 파일 시스템과 비교한다. 마지막으로, 5장에서는 결론과 함께 향후 연구과제에 대해 기술한다.

2. 관련 연구

본 장에서는 기존 NAND 플래시 메모리 전용 파일 시스템의 문제점을 마운트 기법과 회복기법 관점에서 설명한다.

2.1 JFFS2(Journaling Flash File System 2)

JFFS는 LFS(Log-structured File System) 방식의 플래시 메모리 파일 시스템으로서 우수한 안정성으로 인해 널리 사용되고 있다. 버전에 따라 JFFS/JFFS2/JFFS3가 있는데, JFFS2부터 NAND 플래시 메모리에 적합한 파일 시스템으로 개발되었으며, 성능을 향상시키기 위해 JFFS3가 현재 개발[6]중에 있다.

JFFS는 데이타를 로그 형태로 플래시 메모리에 순차적으로 쓰고, 읽기 연산은 로그를 역순으로 검색하여 가

장 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 데이타 압축 기능을 사용한다.

메모리 사용량이 48Bytes에서 16Bytes로 줄었다고는 하지만, 플래시 메모리가 128MBytes일 때, 2^{18}페이지를 필요로 할 수 있기 때문에 jffs2_raw_node_ref를 위한 공간만 2^{22}Bytes, 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.

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.

2.2 YAFFS(Yet Another Flash File System)/ YAFFS2

YAFFS는 NAND 플래시 메모리를 위해 처음으로 개발된 파일 시스템으로 NAND에 최적화 되었으며, 읽기/쓰기/마운트/메모리 소모량 등에서 성능이 우수하다. YAFFS는 소블록 플래시 메모리만을 지원하고, YAFFS2는 YAFFS를 기반으로 대블록 플래시 메모리를 지원할 수 있도록 개발되었다.

YAFFS는 파일을 쓰기 위해 파일의 메타데이타를 저장하는 페이지를 먼저 쓰고, 그 파일에 속한 모든 페이지의 잉여 영역에 공통된 파일 ID를 기록하며, RAM에 저장된 트리를 통해 이들 페이지의 연관 관계를 파악한다. 만일 파일에 속한 어떤 페이지가 더 이상 유효한 값을 갖고 있지 않다면, 그 페이지의 잉여 영역이 'Dirty'로 표시된다. 블록에 속한 모든 페이지가 'Dirty'로 표시된 경우나 플래시 메모리에 가용 공간이 없을 경우, 유효한 페이지를 다른 블록으로 복사한 후 해당 블록을 삭제한다.

YAFFS2는 대부분의 YAFFS의 소스 코드를 상당 부분 공유하며, 정책을 그대로 사용하지만, 유효하지 않은 페이지를 더 이상 'Dirty'로 표시하지 않는다. YAFFS2는 YAFFS와 달리 단조 증가하는 'Sequence Number'를 블록에 저장함으로써 블록이 만들어진 순서를 알 수 있게 하였다. 그 결과 같은 ID를 가진 두 개의 페이지가 존재하더라도 'Sequence Number'에 의해 어느 페이지가 유효하지 않은 페이지인지를 판별할 수 있다.

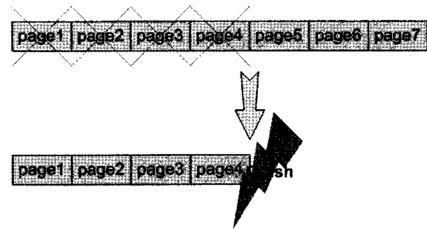


그림 1 파일 수정 중 크래쉬 난 경우

부팅 시 마운트 속도가 매우 빠르고 NAND 플래시 시스템에서는 JFFS2보다 월등히 앞선 성능을 보인다. YAFFS는 마운트 시 파일 시스템 초기화를 위해 스캐닝할 때 잉여영역만 읽기 때문에 JFFS2보다 빠르지만, 다음과 같은 한계가 있다.

YAFFS는 최대 2^{18}개의 파일을 가질 수 있고, 한 파일의 크기는 최대 2^{20}Bytes까지 가능하며, 최대 1GByte의 플래시 메모리를 지원하는 반면, YAFFS2는 8GBytes까지 지원할 수 있다. YAFFS의 가장 큰 문제 중의 하나는 안정성이다. 그림 1과 같이 파일이 수정되는 도중에 파일 시스템이 크래쉬 되면, 수정하기 전 파일의 페이지 1, 2, 3, 4는 삭제되고 페이지 5, 6, 7은 남는다. 따라서, 수정된 페이지 1, 2, 3, 4와 기존의 페이지 5, 6, 7이 파일을 구성하게 되어 파일의 일관성이 깨지게 되고 다음 마운트 시 파일시스템이 마운트가 되지 않는 문제점이 있다. 마지막으로 YAFFS는 마운트 시 플래시 메모리 전체 대신 잉여 영역만을 스캔 함으로써 JFFS2보다는 마운트 시간이 감소했으나, 여전히 플래시 메모리 크기에 비례하여 증가한다.

3. 마운트 및 안정성 기법

본 장에서는 빠른 마운트와 안정성을 제공하기 위해 본 논문에서 개발된 NAND 플래시 메모리 전용 파일 시스템인 NFFiS에서 채택한 마운트 및 안정성 기법에 대해 기술한다. 본 논문에서 구현한 기법은 2KBytes 페이지에 기반하며, MTD(Memory Technology Device) [7]에 논리 페이지 기능을 할 수 있는 Multi-Plane기능을 추가하여 512Bytes 페이지 기반의 NAND 플래시 메모리에서도 동작이 가능하도록 하였다.

그림 2는 NFFiS의 전체 구조를 보여주고 있다. 이 구조는 NAND 플래시 메모리 장치 드라이버를 지원하는 MTD Layer 모듈과 NAND 플래시 메모리 전용 파일 시스템을 지원하는 NFFiS 모듈로 구성된다. MTD Layer는 파일 시스템이 플래시 메모리에 접근할 수 있도록 파티션을 제공하며, 파일 시스템은 파티션을 마운트하여 페이지 읽기, 페이지 쓰기, 블록 삭제 연산을 수행한다.

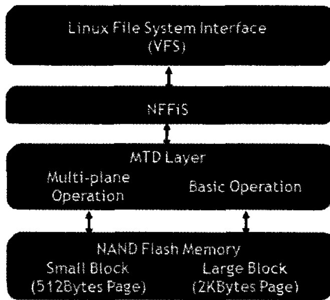


그림 2 제안한 기법이 적용된 NFFiS의 전체 구조

3.1 Multi-Plane

본 논문에서는 MTD에 Multi-Plane기능을 추가함으로써 512Bytes 페이지 크기의 NAND 플래시 메모리도 2KBytes의 논리페이지 파티션을 생성할 수 있게 하였다.

특정 NAND 플래시 메모리는 메모리 셀의 구성이 Multi-Plane으로 되어 있어서 서로 다른 플레인에 속한 여러 페이지에 쓰기 연산, 블록 삭제 연산을 동시에 수행할 수 있다. 이러한 Multi-Plane 연산을 활용하여 512Bytes 페이지 크기의 NAND 플래시 메모리도 2KBytes의 논리페이지 파티션을 생성할 수 있다.

Multi-Plane 형태로 구성된 플래시 메모리는 그 크기에 따라 4개에서 8개의 플레인으로 구성된다. 각 플레인은 1024개의 블록과 528Bytes의 페이지 레지스터를 가진다. 이러한 구성으로 인해 서로 다른 플레인에 속한 다수의 페이지나 블록에 대한 동시 쓰기 연산이나 블록 삭제 연산을 수행할 수 있게 된다.

그림 3에서처럼 첫 번째 블록의 네 페이지에 저장될 데이타는 첫 번째 블록에서부터 네 번째 블록까지의 첫 번째 페이지에 저장된다. 이러한 Multi-Plane 쓰기 연산을 통해 2KBytes 페이지 기반 마운트 기법의 설계가

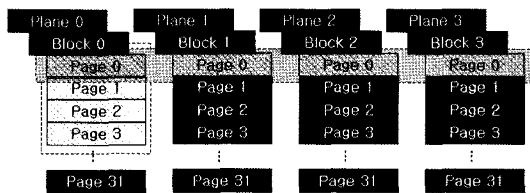


그림 3 Multi-Plane 쓰기 연산

가능하다. 또한, 각 플레인이 가지고 있는 페이지 레지스터를 이용하기 때문에 512Bytes 쓰기 연산을 수행하는 것과 비슷한 시간이 걸리므로, 쓰기 성능이 크게 향상될 수 있다. Multi-Plane을 위해 표 1과 같이 MTD에 Multi-Plane 연산을 추가하였다. Multi-Plane 연산을 지원하는 모든 함수는 Multi-Plane 연산에 적합하게 페이지 크기가 전달되었는지를 조사하기 위해 우선 얼라인먼트를 체크하고, 이상이 없을 경우 512Bytes씩 주소를 증가하며 해당하는 연산을 4번 반복 한다.

3.1.1 MBlock (Multi-Plane Block)

MBlock(Multi-Plane Block)(그림 4)은 Multi-Plane을 이용하여 서로 다른 플레인에 있는 소블록 4개로 구성된 논리 블록이다. MBlock은 32개의 페이지로 구성되며, 각 페이지의 크기는 2KBytes, 각 블록의 크기는 64KBytes이며, Multi-Plane의 삭제 단위이다.

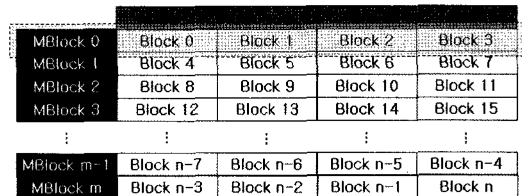


그림 4 MBlock 구조

3.2 잉여 영역(Spare Area)의 데이타 구조

YAFFS는 소블록(Small Block) NAND 플래시 메모리를 위해 개발되었다. 따라서, 페이지 크기가 큰 대블록(Large Block) NAND 플래시 메모리에 적합하도록 기존 YAFFS의 각 필드를 수정하여 NFFiS에 적용하였다.

NAND 플래시 메모리의 각 페이지마다 데이타 영역에는 파일 데이타가 저장되고, 잉여 영역에는 NAND 플래시 메모리 정보, 데이타 영역에 저장된 파일 데이타를 관리하기 위한 메타 데이타인 Tag, 페이지 데이타의 에러정정코드(Error Correction Code, ECC)가 저장된다. Tag는 파일 시스템이 지원할 수 있는 파일의 개수와 파일의 크기, 가용한 갱신 횟수를 최대화하면서 램 사용량을 최소화하도록 설계되었고, 잉여영역에 저장되는 Tag의 구조는 그림 5와 같이 16Bytes로 구성된다.

잉여 영역에 저장되는 구조체의 각 항목은 다음과 같

표 1 Multi-Plane 연산

연산	기능
nand_read_ecc_m	각 플레인에서 플래시 메모리의 페이지를 동시에 512Bytes씩 읽음
nand_write_ecc_m	각 플레인마다 512Bytes씩 할당하여 동시에 플래시 메모리에 씀
nand_erase_m	Multi-Plane 블록 삭제
nand_read_oob_m	각 플레인에서 플래시 메모리의 잉여 영역을 동시에 읽음
nand_write_oob_m	각 플레인에 플래시 메모리의 잉여 영역을 동시에 씀

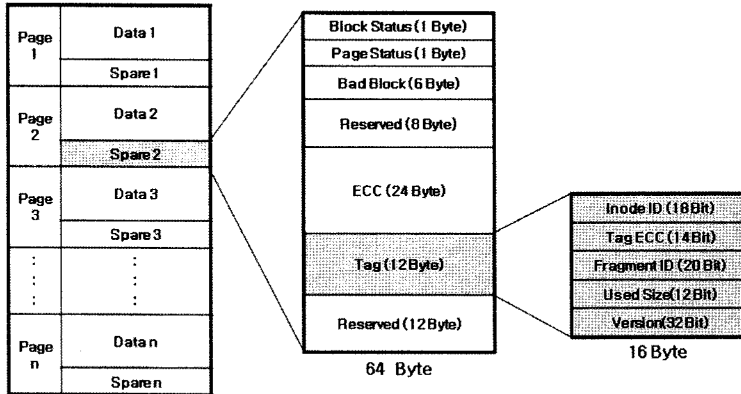


그림 5 잉여 영역에 저장되는 Tag의 구조

이 이용된다.

• Block Status

Block Status는 해당 페이지가 속해 있는 블록이 배드 블록임을 표시하는데 사용된다. NAND 플래시 메모리는 NOR 플래시 메모리와 달리 배드 블록을 포함한 상태로 출시될 수 있을 뿐만 아니라, 사용하는 과정에서 삭제 연산이 누적되어 입출력 에러가 발생하는 블록이 생길 수 있다. 따라서 배드 블록이 더 이상 데이타 입출력에 사용되지 못하도록 하는 기능이 필수적인데, 구현한 파일 시스템에서는 입출력 에러가 발생하는 블록이 생기면 그 블록의 첫 번째와 두 번째 페이지 잉여 영역(Spare1과 Spare2)의 Block Status에 이를 표시하고, 해당 블록이 더 이상 데이타 입출력에 사용되지 않도록 한다. 출시할 당시의 배드 블록 표시는 배드 블록 표시 위치에 기록되어 있으므로 이를 이용하여 배드 블록을 감지할 수 있다.

• Page Status

Page Status는 무효화 된 페이지를 표시하는데 사용된다. 어떤 파일의 데이타가 갱신되면 다른 페이지에 새로이 저장되고 이전 페이지에 저장된 데이타는 무효화된다. 구현한 파일 시스템은 무효화된 페이지를 Page Status 필드를 이용하여 표시하고, 모든 페이지가 무효화된 블록은 가비지 컬렉션 기능을 이용하여 삭제하고, 데이타가 다시 저장될 수 있다.

• ECC(Error Correction Code)

페이지에 저장된 데이타의 ECC를 저장하는데 사용된다. NAND 플래시 메모리는 페이지에 데이타를 저장하는 도중, 에러가 발생할 가능성이 있기 때문에 구현한 파일 시스템은 페이지 데이타의 ECC값을 잉여 영역에 저장하여 페이지 데이타의 무결성을 검사한다.

• Tag

12Bytes로 이루어진 Tag에는 페이지에 저장된 데이타에 대해 다음과 같은 정보가 저장된다.

· Inode ID

페이지에 저장된 데이타를 소유하는 파일의 ID를 나타내며, 리눅스 가상 파일 시스템의 Inode 번호가 저장된다. Inode ID를 위해 18Bits가 할당되며, 최대 218개의 파일을 지원할 수 있다.

· Fragment ID

페이지에 저장된 데이타의 파일 내 오프셋을 나타낸다. Fragment ID 값이 0이면, 그 페이지에는 파일에 대한 메타데이타를 저장하는 Inode Header의 데이타가 저장되어 있음을 나타내고, Fragment ID 값이 n(n>0)이면, 페이지에는 파일의 n번째 페이지가 저장되어 있음을 나타낸다. Fragment ID로 20Bits가 할당되었기 때문에, 하나의 파일은 최대 2^{20}개의 페이지를 소유할 수 있다. 대블록 NAND 플래시 메모리는 페이지 크기가 2KBytes이므로 파일 시스템은 최대 2GBytes인 파일을 지원할 수 있다. 또한, 2^{18}개의 파일을 지원할 수 있으므로, 최대 512TBytes 크기의 플래시 메모리 파일 시스템을 지원할 수 있다.

· Used Size

페이지에 저장된 데이타의 Byte 수를 표시한다. 특정 파일에서 마지막 페이지를 제외하고는 NAND 플래시 메모리 페이지 크기와 동일한 값을 가지게 된다. 페이지 크기가 2KBytes이므로 12Bits가 할당된다.

· Version

갱신된 페이지의 생성된 순서를 구분하기 위한 것으로 마운트와 저널링에서 사용되며, 갱신이 완료된 후 1씩 증가한다. Version에 32Bits가 할당되었기 때문에, 각 파일은 232번까지 수정될 수 있다.

· Tag ECC

14Bits가 할당된 Tag ECC는 Tag에 대한 에러 정정 코드가 저장되는 장소로서 잉여 영역에 Tag를 저장

할 때 에러 발생여부를 검사하는데 사용된다.

3.3 Mount 기법의 구성 요소

본 논문에서 제안한 기법의 주요 구성 요소에는 마운트 시 스캔 되는 잉여 영역의 범위를 최소화하기 위해 ExBlock이라는 새로운 블록 개념과 이를 관리하는 SNode와 SNode 생성을 요청하는 VNode 등이 있으며, 그 구조는 그림 6과 같다.

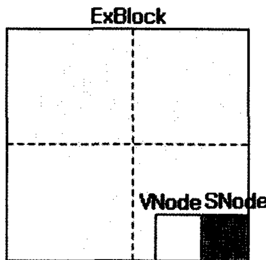


그림 6 ExBlock, SNode, VNode 구조

3.3.1 ExBlock(Extended Block)

ExBlock은 연속적인 몇 개의 물리적 블록을 하나로 묶은 논리적 확장 블록으로서 삭제 단위가 된다. ExBlock의 크기는 NAND 플래시 메모리의 물리적 구성에 따라 다르며, 식 (1)을 적용하여 크기를 구한다.

$$\text{페이지 정보 크기} * \text{블록 페이지 개수} * \text{블록 개수} \le \text{NAND 플래시 메모리 페이지 크기} \quad (1)$$

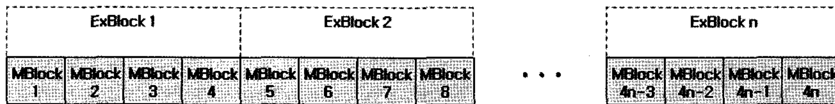
페이지 정보의 크기는 12Bytes이며, 이러한 페이지 정보는 본 3.3.2절에서 설명할 SNode에 저장된다. 블록 페이지 개수는 NAND 플래시 메모리의 블록 당 페이지 개수이다. 블록 개수는 식 (1)의 NAND 플래시 메모리 페이지 크기를 초과하지 않으면서, 가장 큰 2의 배수로 정해 진다. 따라서, 페이지 정보를 가질 수 있는 NAND 플래시 메모리 페이지의 크기가 커지면, 더 큰 ExBlock을 가질 수가 있어 마운트 속도를 더욱 더 빠르게 할 수 있다.

그림 7은 식 (1)을 이용하여 구성된 NFFiS의 소블록 및 대블록 ExBlock 구조를 보여준다. 소블록(그림 7(a))에서는 4개의 MBlock, 즉 16개의 블록이 하나의 ExBlock으로 구성되며, 대블록(그림 7(b))에서는 2개의 블록이 하나의 ExBlock블록으로 구성된다.

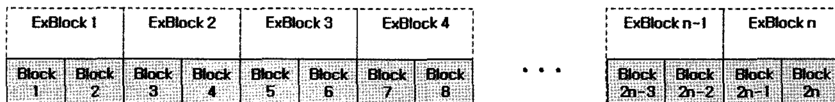
3.3.2 SNode(Snapshot Node)

SNode는 ExBlock에 존재하는 모든 Fragment의 메타 데이타가 저장된 테이블로서 ExBlock의 마지막 페이지에 저장되며, 그 구조는 그림 8과 같다.

파일 시스템은 마운트 시 ExBlock 안에 존재하는 모



(a) 소블록의 ExBlock 구조



(b) 대블록의 ExBlock 구조

그림 7 ExBlock 구조

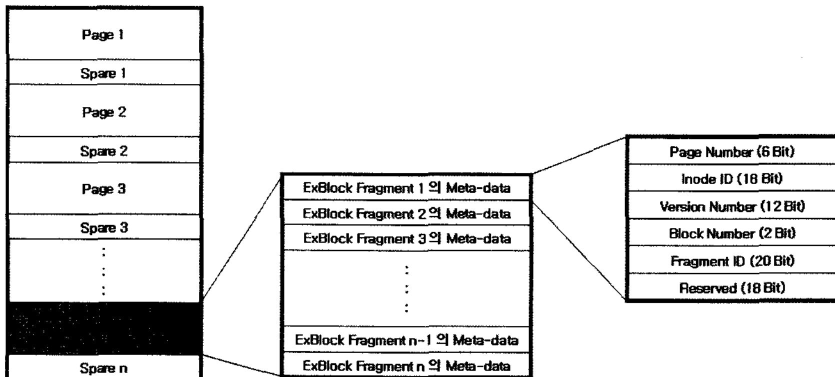


그림 8 SNode 구조

표 2 SNode 테이블 요소

연산	기능	크기
Page Number	블록 내에서 해당 Fragment의 위치	6 Bits
Inode ID	Fragment가 속한 Inode의 ID	18 Bits
Version Number	Fragment의 버전	32 Bits
Block Number	ExBlock내에서 해당 블록의 위치	2 Bits
Fragment ID	Inode 내에서 Fragment의 순서	20 Bits
Reserved	미래를 위해 예약된 공간	18 Bits

든 Fragment들의 잉여 영역을 스캔 하지 않고, SNode 테이블만을 참조하여 파일 시스템을 구성한다.

3.3.3 VNode(Victim Node)

ExBlock에서 데이타 저장을 위해 마지막으로 할당되는 페이지로서 VNode에 쓰기를 마친 후에 파일시스템은 SNode 큐에 생성할 SNode의 해당 ExBlock 번호를 입력한다. 이는 쓰기연산 동안에는 SNode를 생성하지 않고 Idle한 시간에 SNode를 생성함으로써 SNode의 생성으로 인한 쓰기 지연 시간을 최소화한다.

3.4 마운트 시 플래시 메모리를 스캐닝 하는 방법

본 연구에서 제안한 마운트 기법은 그림 9의 ①에서와 같이 NAND플래시 메모리를 스캐닝할 때, ExBlock의 메타 데이타를 저장하고 있는 SNode만 읽는다. 그림 9의 ②에서와 같이 SNode가 존재하지 않는 ExBlock의 경우, 그림 9의 ③에서처럼 ExBlock안에 존재하는 모든 잉여 영역을 스캔 한다. 이처럼 SNode의 스캔만으로 ExBlock에 존재하는 모든 데이타의 정보를 알 수 있어서 마운트 시간을 크게 줄일 수 있다. 또한, ExBlock내 Block의 수를 플래시 메모리의 페이지 크기에 맞춰서 조절함으로써, 마운트 시간이 플래시 메모리 크기에 선형적으로 증가하는 것을 방지함으로써 마운트 시간을 줄일 수 있다.

3.5 SNode 생성

그림 10에서 페이지 190이 ExBlock1의 VNode로서 SNode 생성을 요청한다. VNode를 쓰고 난 후, 다음 데이타 페이지인 ExBlock2의 페이지 192를 쓰기 전에

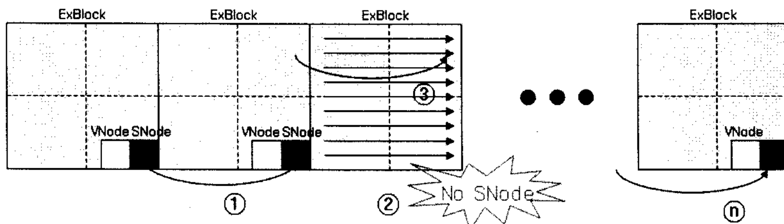


그림 9 마운트 시 스캐닝 과정

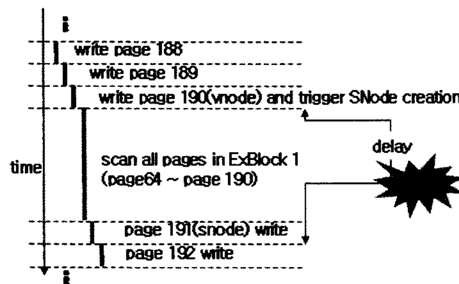


그림 10 쓰기 중 SNode 생성의 문제

현재의 ExBlock1을 스캔하여 SNode가 생성된다면, SNode가 생성될 때까지 다음 쓰기 작업은 멈추게 된다. 이와 같이 쓰는 도중 ExBlock 내 페이지가 전부 채워질 때마다 해당 ExBlock에 대한 SNode를 생성하게 되면, 쓰기 성능이 크게 떨어지게 된다.

따라서, 쓰기 동안에는 SNode를 생성하지 않고, 쓰기를 마친 페이지가 VNode라면 SNode 큐에 해당 ExBlock 번호를 추가한다. Idle한 시간에 SNode 큐에서 추가된 ExBlock 번호를 하나씩 선택하여 SNode를 생성한다. SNode에는 ExBlock내 모든 페이지의 태그를 순차적으로 읽어 필요한 정보를 추출하여 저장한다.

NAND 플래시 메모리의 Idle한 시간을 파악하기 위해, 마운트 시 슈퍼블록이 생성되면, 타이머 커널 스레드를 생성한다. 타이머 커널 스레드는 Sleep하고 있다가 2초 이상 Idle하게 되면 깨어나, SNode를 생성한다. 이는 디스크가 2초 동안 Idle하면, 다시 4초간 Idle일 확률이 95% 이상이라는 실험[8]에 기초하고 있다. 따라서, 쓰기 지연을 줄이기 위해 디스크가 2초 동안 Idle하면, SNode 큐에서 생성해야 할 SNode가 있는 지를 검사하고 있으면 만들어야 할 ExBlock 번호를 SNode 큐에서 읽어 SNode를 생성한다.

3.6 저널링

3.6.1 Fragment Tree

제안한 기법에서 논리 페이지 주소는 한 파일 내에서 Fragment의 오프셋을 나타내는 Fragment ID가 된다. Inode의 Fragment를 삭제 또는 수정하고자 할 때, 관련된 Fragment가 실제로 플래시 메모리 상의 어디에 위치하는지를 알아야 하는데, 제안한 기법에서는 Fragment Tree를 이용하여 Fragment ID를 플래시 메모리의 페이지 주소로 변환한다. Fragment Tree는 각 Inode 별로 데이타가 저장되어 있는 페이지들을 계층적인 트리 구조를 이용하여 관리함으로써 Fragment 데이타가 저장되어 있는 페이지 주소를 빠르게 찾아낼 수 있도록 하며 주소변환 과정에서 사용되는 메모리 사용을 최소화 한다. 일반적으로 보조기억장치에서의 탐색은 시간적인 부하가 많이 걸리기 때문에 탐색을 쉽게 하기 위해 파일과는 별도로 인덱스를 만들어 사용한다. 인덱스가 커질 경우, 인덱스 역시 보조기억장치에 저장하는데 보조기억장치는 상대적으로 느리므로 보조기억장치의 접근 횟수를 최대한 억제시켜야 한다. 인덱스에 대한 접근 횟수를 줄일 수 있도록 Tree의 높이를 낮추기 위해 본 논문에서는 YAFFS와 같이 B+-Tree를 사용한다.

Fragment Tree 구조를 사용하여 파일을 관리할 때, 하나의 내부 노드는 64Bytes의 크기를 가지며, 16개의 페이지 주소를 관리할 수 있다. 따라서, 2KBytes 크기의 페이지를 가진 128MBytes 용량의 NAND 플래시 메모리를 사용할 경우, 최대 512KBytes의 메모리 공간이 말단 노드를 위해 필요하다. 여기에 내부 노드와 Inode 구조체를 사용하기 위한 약간의 메모리 공간이 추가로 요구된다.

• Fragment Tree 종류

· 말단 노드 계층

말단 노드는 16개의 16Bits 페이지 주소를 포함하고 있다. 말단 노드에 기록되어 있는 페이지 주소는 해당 Fragment가 플래시 메모리 상에서 저장되어 있는 페이지의 실제 주소가 된다. Fragment ID의 최하위 4Bits 값은 말단 노드에 기록되어 있는 16개의 페이지 주소들 중에서 해당하는 주소를 선택하기 위한 인덱스로 사용된다.

· 내부 노드 계층

내부 노드는 말단 노드를 묶어서 관리하며, 하나의 내부 노드는 최대 16개의 말단 노드를 관리한다. Fragment Tree는 Fragment ID의 상위 4Bits의 값을 인덱스로 사용하여 하위 노드로 이동한다.

3.6.2 Fragment Version

본 논문에서 제안한 기법은 마운트 시 파일에 대한 Fragment Tree를 구성한다. 저널링을 위해 트리의 말단 노드에 Fragment의 정보가 리스트로서 유지된다. 그림 11은 마운트 시 생성되는 Fragment Tree이다. 파일이 최초로 생성되면, Version은 0부터 시작되고, 파일이 Close나 Flush될 때마다 해당 Fragment의 Version이 1씩 증가한다. 또한, 파일이 Close되지는 않았지만, 파일에서 업데이트가 이루어진 후 일정 시간이 경과하면 Version이 증가한다. 파일의 수정이 완료되면, Fragment ID가 0번인 헤더의 Version도 1 증가한다.

3.6.3 저널링 정책

본 논문에서 제안하는 저널링 정책은 마운트 정책에서 제안한 ExBlock에 적합하도록 설계되었다. ExBlock의 SNode에 저장된 내용은 ExBlock이 삭제되기 전까지 NAND 플래시 메모리 스캔 시 정보를 읽어 와서 그림 11과 같이 Log Lists에 쌓이게 된다. 마운트가 종료된 후에는 유효한 Fragment만 가지기 위해 표 3과 같은 저널링 정책을 적용한다. 스캔 도중 Fragment ID 0번이 갱신되면 Fragment를 정리하는 정책과 스캔이 마친 뒤에는 남은 Fragment를 정리하는 스캔 종료 후 정책을 적용하여 유효한 Fragment만을 남기게 된다.

그림 12는 스캔할 때 Fragment를 이용하여 저널링을 수행하는 예이다. 그림 12(a)는 하나의 파일에서 파일을 구성하는 각 Fragment 마다 Version들의 리스트를 유지하고 있는 상태이다. Fragment ID 0번은 파일의 헤더로서 파일 변경이 정상적으로 종료된 후에만 Version이 업데이트 된다. 이 예에서는 Fragment ID 0번의

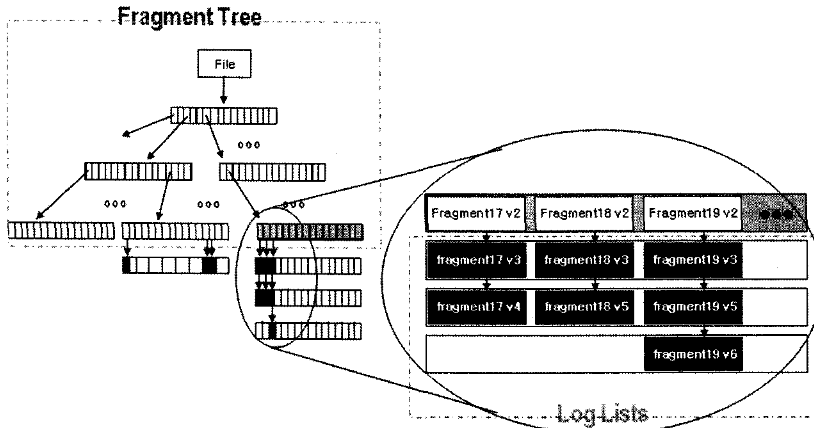


그림 11 마운트 시 생성되는 Fragment Tree

표 3 저널링 정책

[스캔 시 Fragment ID 0번 Version 갱신 후 정리 정책] • Fragment ID 0번 Version ≥ Fragment Version 경우, - Fragment 중 최신 Version의 Fragment만 유지 - Fragment 중 Version이 같은 Fragment가 있으면, Version이 작은 Fragment는 삭제 • Fragment ID 0번 Version < Fragment Version 경우, - Fragment ID 0번 Version 보다 큰 Fragment들은 유지 **[스캔 종료 후 정책]** • Fragment ID 0번 Version < Fragment Version 경우, - Fragment ID 0번 Version 보다 큰 Fragment들은 삭제

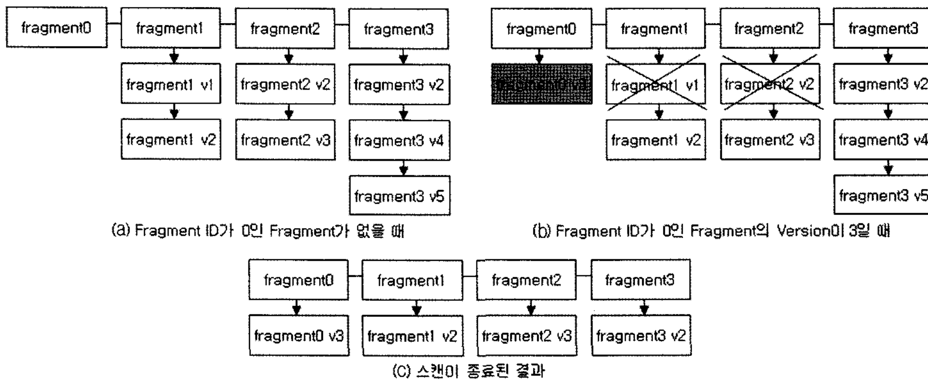


그림 12 저널링 동작 예

Version은 3이 된다.

Fragment ID 0번에 새로운 Version이 입력되면 무효화된 Fragment들은 정리된다. 그림 12(b)는 Fragment ID 0번의 Version이 3으로 변경된 후 Fragment들이 정리된 상태이다. Fragment 1번에 Version이 1과 2인 페이지가 있는데, Fragment 0번의 Version이 3이면 Version 3까지가 완료된 것이므로 Fragment 1번에서는 Version이 3 이하인 것 중에서 가장 최근 것인 Version 2만 남겨두고, Version 1에 해당하는 페이지를 지운다. Fragment 2번의 경우, Version 3이 가장 최근에 변경된 파일을 가지고 있으므로 Version 2가 삭제된다. Fragment 3번과 같이 Version 2와 Version 4, Version 5를 가지고 있는 경우, Version 4와 Version 5가 확실히 쓰기가 완료된 자료인지 여부를 확인할 수 없기 때문에 Version 2, 4, 5번을 모두 유지한다. 그림 12(c)은 Fragment 0번의 Version이 4 또는 5로 변경되지 않은 채 스캔이 종료된 경우, Frag-

ment 3의 Version 4, 5는 에러가 발생했거나 쓰기를 완료하지 못한 상태에서 발생한 Fragment이므로 해당 Fragment가 삭제된 후의 결과이다.

4. 성능 평가

본 장에서는 성능 평가를 위해 본 논문에서 제안한 NFFiS와 기존의 NAND 플래시 메모리 전용 파일 시스템인 JFFS2, YAFFS, YAFFS2를 동일한 환경에서 테스트하여 그 결과를 비교한다.

4.1 테스트 환경

성능 평가를 위해 구축한 테스트 환경은 크게 Host 컴퓨터와 Target 보드로 구성된다. Host 컴퓨터(표 4 참조)는 Target 보드의 루트 파일 시스템을 제공하는 시스템으로서 NFS(Network File System)를 이용하여 Target 보드가 부팅될 수 있도록 지원한다.

표 4 Host 컴퓨터 사양

사양	CPU	Pentium III 933 MHz
	RAM	512MBytes
	NIC	Intel PRO/100+Management Adapter
운영체제		Hancom Linux 4.0(Linux Kernel 2.6.6)

리눅스 포팅을 위한 Target 보드로서는 ARM(ARM9)과 DSP(C55x)로 구성된 OMAP 5912 OSK(OMAP Start Kit)가 사용되었다(표 5 참조).

표 5 Target 보드 사양

사양	Model	OMAP5912 OSK
	MCU	ARM926Ej-Sid(wt) rev 3 (v51)
	RAM	32MBytes DDR SDRAM
	Flash	32MBytes(NOR)
	기타	NAND Flash Test Board RS-232 Serial Port, 10Mbps Ethernet Port, USB Port
Boot Loader		u-boot 1.1.1
Linux Kernel		Linux Kernel2.6.10 for OMAP
Root File System		NFS

성능 평가 실험에 사용된 NAND 플래시 메모리의 형번호와 재원은 표 6과 같다. 대블록인 NAND 플래시 메모리는 64MBytes 크기의 파티션으로 나누어 성능을 측정한다.

표 6 NAND 플래시 메모리 재원

형 번호	용량	페이지 크기
삼성 K9F1208U0M	64MBytes	(512 + 16)Bytes
삼성 K9F2G08U0M	256MBytes	(2048 + 16)Bytes

4.2 마운트 성능 분석

소블록과 대블록 플래시 메모리에서 마운트 시간을 동일한 방법으로 측정하여 이를 비교한다.

4.2.1 소블록

파일 시스템을 마운트하는데 걸리는 시간은 플래시 메모리에 저장되어 있는 파일의 크기와 파일의 개수에 크게 영향을 받는다. 그림 13은 플래시 메모리에 존재하는 파일의 크기를 변화시켰을 때, 각 파일 시스템의 마운트 시간을 측정한 결과이며, 그림 14는 플래시 메모리에 존재하는 파일의 개수를 변화시켰을 때의 결과이다.

세 개의 파일 시스템은 저장된 파일의 크기와 파일의 개수에 비례하여 마운트 시간이 일제히 증가하였는데, JFFS2는 다른 두 파일 시스템에 비해 급격하게 증가하는 것으로 나타났다. NFFiS는 ExBlock의 SNode만 읽어 마운트를 하기 때문에 파일의 크기와 개수에 크게 영향을 받지 않고 있다. 플래시 메모리에 그림 14와 같이 50MBytes가 저장되었을 경우, NFFiS는 YAFFS에 비해 2.46배, JFFS2에 비해서는 14.31배 정도 좋은 성능을 보였다.

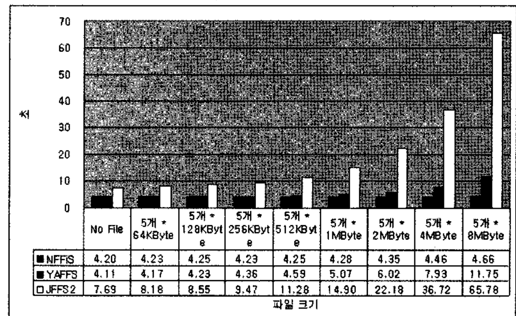


그림 13 파일 크기에 따른 마운트 성능

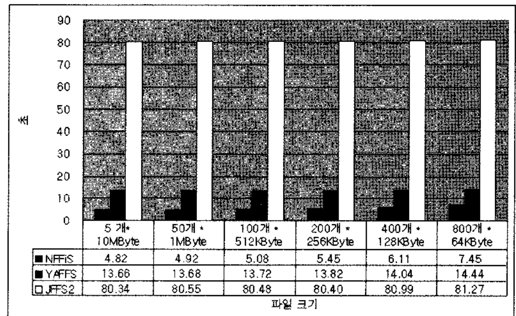


그림 14 파일 개수에 따른 마운트 성능

4.2.2 대블록

그림 15는 플래시 메모리에 저장되어 있는 파일의 크기를 변화시키고, 그림 16은 파일의 개수를 변화시켰을

때, 각 파일 시스템의 마운트 시간을 측정한 결과이다.

대블록 플래시 메모리에서도 소블록 플래시 메모리와 같이 NFFiS의 성능이 가장 좋고, JFFS2가 성능이 가장 낮다. NFFiS는 YAFFS2에 비해 최대 1.7배, JFFS2에 비해 16.52배 정도 좋은 성능을 보였다.

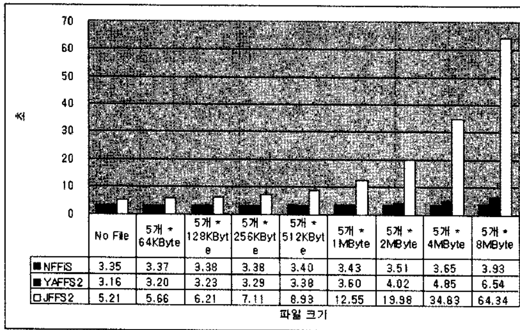


그림 15 파일 크기에 따른 마운트 성능

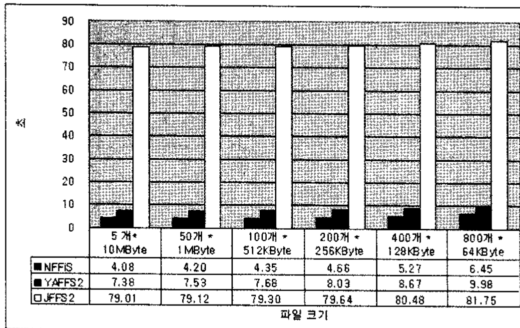


그림 16 파일 개수에 따른 마운트 성능

4.3 안정성 분석

안정성(Robustness) 테스트와 정전 테스트를 수행하여 본 논문에서 제안한 저널링 방법이 안정성 확보에 문제가 없는지를 테스트 하고, 기존의 YAFFS 및 JFFS2와 비교하였다.

4.3.1 안정성 테스트

안정성 테스트는 외부로부터의 영향이 없는 상태에서 오랜 시간 동안 시스템에 많은 부하를 주었을 때, 시스템이 오작동하지 않고 정상적으로 작동 하는지를 평가하는 것이다. 안정성 테스트에 사용될 시나리오는 표 7과 같다.

YAFFS를 이용하여 안정성을 테스트할 경우, NAND 플래시 메모리 크기보다 큰 파일을 복사하는 과정에서 파일 시스템에 오류가 발생하였고, 복구가 제대로 이루어지지 않았다. NAND 플래시 메모리 크기보다 큰 파일을 복사할 때, 에러가 제대로 처리되지 않아 시스템 오류가 발생한 것인데, YAFFS의 파일 수정 정책의 문제점으로 인해 파일의 일관성이 깨지면서 복구가 제대로 이루어지지 않은 것이다.

YAFFS 파일 시스템에 본 논문에서 제안한 저널링 기법을 적용한 경우는 JFFS2와 같이 오류가 발생하지 않았고, 또한 24시간 안정성 테스트 시나리오를 수행한 경우에도 오류 없이 동작하였다.

안정성 테스트 결과, 본 논문에서 제안한 NFFiS의 저널링 기법은 JFFS2와 마찬가지로 안정성에 문제가 없었다.

4.3.2 정전 (Power Failure) 테스트

NAND 플래시 메모리는 휴대폰, PDA, 디지털 카메라, MP3 플레이어 등과 같은 휴대용 기기에 많이 사용

표 7 안정성 테스트 시나리오

시나리오 순서	내 용
1	하위 디렉터리를 path_max(4096Bytes)만큼 재귀적으로 생성한다
2	최상위 디렉터리의 파일을 각각 하위 디렉터리에 파일을 복사한다
3	하위 디렉터리의 소프트 링크를 최상위 디렉터리에 생성한다
4	하위 디렉터리의 파일 하드 링크를 최상위 디렉터리에 생성한다
5	다양한 크기의 파일을 생성하고 복사, 삭제, 압축, 권한 변경을 반복 실시하여 NAND 플래시 메모리의 용량을 가득 채운다
6	1~5번의 작업을 10회 반복 후, 파일 시스템을 언마운트 한다
7	다시 파일 시스템을 마운트 한다
8	1~7번의 과정을 반복한다

표 8 정전 테스트 통과 기준

단위 시험 방법	통과 기준
마운트 작동 시험	파일 시스템의 정상 등록
응용 프로그램을 통한 파일 시스템 시험	파일, 디렉터리, 소프트 링크, 하드 링크의 손실 여부
	파일, 디렉터리, 소프트 링크, 하드 링크의 생성, 읽기, 삭제, 복사, 이동의 정상 동작 여부

되고 있기 때문에 갑작스런 정전이 자주 일어날 수 있다.

표 8의 안정성 테스트 시나리오 수행 중 임의의 시간에 전원을 차단함으로써 시스템이 오작동을 하는지를 검사한다. 정전 테스트의 통과 기준은 표 12와 같다.

YAFFS의 경우, 파일의 수정 중 정전이 일어날 경우, 정상적으로 파일 시스템을 마운트하지 못했다. 본 논문에서 제안한 NFFiS의 저널링 기법은 JFFS2와 같이 파일의 수정 중 정전이 일어나도 정상적으로 파일 시스템을 마운트 할 수 있었으며, 응용 프로그램을 통한 파일 시스템 시험도 정상적으로 통과하였다. 그러나 NAND 플래시 메모리 크기의 절반 이상이 되는 큰 용량 파일을 수정할 때, 정전이 일어나면 파일의 일관성이 깨지며 그 파일을 폐기(discard) 시켰다. 이것은 크기가 NAND 플래시 메모리 용량의 절반 이상인 파일이 수정되면, 이전의 Fragment ID 0번 페이지가 먼저 삭제된다. 또한, 이전에 해당 파일이 있던 자리 중 일부가 재사용되기 때문에 안전하게 종료된 Fragment들이 있던 블록들이 삭제 된다. 따라서, 정전이 일어나면 다음 마운트 시 Fragment ID 0번이 없기 때문에 모든 해당 Fragment들이 유효한지를 확인할 수가 없어 모두 무효한 Fragment가 된다. 이처럼 저널링 정책으로 인하여 스캔 종료 후에는 해당 Fragment가 삭제되어, 복구는 안 되지만, 정전 후 재부팅 시 파일 시스템은 성공적으로 마운트되어 사용할 수 있었다.

4.4 소블록의 읽기/쓰기 성능 분석

4.4.1 읽기 성능

그림 17은 플래시 메모리에서 1MBytes의 파일을 읽을 때, 벤치마킹 프로그램[9]의 버퍼 크기에 따른 읽기 성능을 나타낸 것이다. 각 파일 시스템은 버퍼의 크기에 따라 성능이 조금씩 다르게 나타났지만, 이 차이는 미미하여 소블록 플래시 메모리에 대한 이들의 읽기 성능은 거의 비슷하다고 할 수 있다.

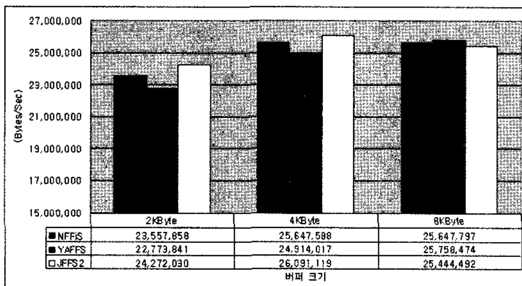


그림 17 읽기 성능

4.4.2 쓰기 성능

그림 18은 플래시 메모리에서 1MBytes의 파일을 쓸 때, 벤치마킹 프로그램의 버퍼 크기에 따른 쓰기 성능을 나타낸 것이다. 본 연구에서 구현한 파일 시스템은 YAFFS에 비해 72%정도, JFFS2에 비해서는 14%~56% 정도 좋은 성능을 보였다. 이러한 성능 향상은 본 연구에서 구현한 NFFiS가 MTD Layer에서 Multi-Plane 연산을 지원하기 때문이다. JFFS2의 경우는 버퍼 크기를 4096Bytes로 설정했을 때부터 성능이 좋아지는데, 이는 JFFS2가 Write-Buffering 정책[3]을 사용하기 때문이다.

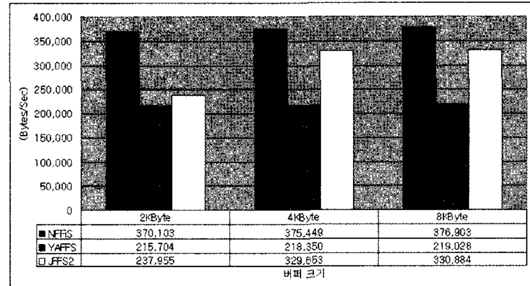


그림 18 쓰기 성능

5. 결론 및 향후 연구

비 휘발성의 특성을 가진 반도체 저장 매체인 NAND 플래시 메모리는 기존의 자기 저장 매체에 비해 집적도가 높고, 적은 소비 전력으로 구동이 가능하다. 또한, 읽기 및 쓰기 성능이 좋으며, 온도와 충격, 진동, 자기 등에 대해 강한 내구성을 가지고 있기 때문에 휴대용 이나 내장형 기기의 저장 매체로 매우 적합하다. 이에 따라 대용량 멀티미디어 데이타에 대한 수요가 증가하면서 NAND 플래시 메모리의 사용이 더욱 증가할 것으로 예상된다.

NAND 플래시 메모리는 데이타를 기록할 공간을 미리 삭제하는 과정을 거쳐야 하며, 그 횟수가 제한적 이어서 NAND 플래시 메모리 전용 파일 시스템이 필요하다. 대표적인 것으로 JFFS2와 YAFFS가 있는데, JFFS2는 안정적이지만 마운트 속도가 상당히 느리고, 읽기 및 쓰기 등의 메모리 접근 성능도 약간 떨어진다. 반면, YAFFS는 JFFS2에 비해 상대적으로 읽기, 쓰기 속도와 마운트 속도가 빠르고 램 소비량도 적지만, 파일 수정 시 안정성에 문제가 있다.

본 논문에서는 YAFFS를 기반으로 설계된 파일 시스템인 NFFiS가 지원하는 NAND 플래시 메모리의 특성을 고려한 빠른 마운트와 안정성 기법에 대해 설명하였다. 빠른 마운트를 위해 마운트 시 스캔되는 영역이 최소화될 수 있도록 연속적인 몇 개의 물리적인 블록을 하나로 묶어 논리적인 확장 블록으로서 ExBlock이라는

새로운 블록 개념, ExBlock에 존재하는 모든 페이지를 관리하는 테이블을 가진 SNode, 그리고 SNode 생성을 요청하는 VNode를 제안하였다. 마운트될 때 SNode만을 스캔함으로써 마운트 시간을 줄일 수 있으며, Fragment Version 정책을 추가하여 안정성을 높였다.

본 논문에서는 NFFiS와 JFFS2, YAFFS, YAFFS2 등의 NAND 플래시 메모리 전용 파일 시스템과 성능을 비교하였다. NFFiS의 마운트 성능은 파일 개수에 따른 성능에서 소블록 NAND 플래시 메모리의 경우, YAFFS와 JFFS2에 비해 각각 최대 2.46배, 최대 14.31배의 성능 향상을 보였다. 대블록 NAND 플래시 메모리의 경우, YAFFS2와 JFFS2에 비해 각각 최대 1.7배, 최대 16.52배 정도의 성능 향상을 보였다. 본 연구에서 구현한 NFFiS는 다양한 테스트 시나리오에서 안정성이 높은 것으로 알려진 JFFS2와 동등한 수준의 안전성을 보였다. 또한, 소블록 NAND 플래시 메모리의 경우, Multi-Plane 연산의 추가로 쓰기 성능에서 YAFFS보다 72% 정도, JFFS2보다 14~56% 정도 우수한 성능을 보였다.

본 논문에서 제안한 NFFiS는 기존의 YAFFS보다 램을 더 필요로 하기 때문에 전력이 더 소비되므로 램 요구량을 줄일 수 있는 방안이 요구된다. 또한, 플래시 메모리에 쓰기가 수행 중일 때, 가용 공간이 없을 경우에도 가비지 컬렉션이 수행되므로 쓰기 성능에 악영향을 줄 수 있다. 따라서, Idle한 시간에 가비지 컬렉션을 수행하는 기능이 추가되어야 한다. 마지막으로, 현재 Multi-Plane에서 동시에 연산이 수행되는 네 블록 중 한 개의 블록이 배드 블록이어도 네 블록 모두 배드 블록 처리가 된다. 이는 플래시 메모리의 이용률 저하의 원인이 되기 때문에 효율적인 배드 블록 관리기법이 개발되어야 한다.

참 고 문 헌

[1] Intel Corporation, "3 Volt Synchronous Intel Strata Flash Memory," http://www.intel.com/.

[2] Samsung Electronics, "NAND Flash Memory," http://www.sec.co.kr/.

[3] JFFS2, http://www.linux-mtd.infradead.org/doc/jffs2.html/.

[4] Aleph One Company, "Yet Another Flash Filing System," http://www.aleph1.co.uk/yaffs/.

[5] Uresh Vahalia, "Unix Internals," Prentice Hall, January 1996.

[6] JFFS3, http://www.linux-mtd.infradead.org/doc/jffs3.html/.

[7] MTD, "Memory Technology Device (MTD) subsystem for Linux," http://www.linux-mtd.infradead.org/.

[8] T. Blackwell, J. Harris and M. Seltzer, "Heuristic Cleaning Algorithms in Log-Structured File Systems," Proceedings of the 1995 Winter Usenix, January 1995.

[9] 박상오, 김성조, "리눅스 기반의 NAND 플래시 메모리 파일 시스템에 대한 성능 측정 도구 설계", 제32회 추계학술발표회 논문집, Vol.32, No.2(1), pp. 547-549, 2005.

[10] D. Woodhouse, Red Hat, Inc. "JFFS: The Journaling Flash File System," Ottawa Linux Symposium, 2001.

[11] 박상호, 안우현, 박대연, 김정기, 박승민, "플래시 메모리를 위한 파일 시스템 구현," 정보과학회논문지 : 컴퓨팅의 실제, Vol.7, No.5, pp.402-415, 2001.

[12] Samsung Electronics, "NAND Flash Memory & SmartMedia Data Book," 2003.

[13] Daniel P. Bovet, Marco Cesati, "Understanding the LINUX KERNEL," O'Relly, November 2005.

[14] Chanik Park, Jeong-Up Kang, Seon-Yeong Park, and Jin-Soo Kim, "Energy-Aware Demand Paging on NAND Flash-based Embedded Storages," International Symposium on Low Power Electronics and Design, 2004.

박 상 오

2005년 중앙대학교 컴퓨터공학과(공학사). 2007년 중앙대학교 컴퓨터공학과(공학석사). 2007년~현재 중앙대학교 컴퓨터공학과(박사과정). 관심분야는 NAND 플래시 메모리 파일 시스템, 임베디드 시스템

김 성 조

1975년 서울대학교 응용수학과(공학사) 1977년 한국과학기술원 전산과(이학석사). 1977년~1980년 ADD(연구원). 1980년~현재 중앙대학교 컴퓨터공학부 교수. 1987년 Univ. of Texas at Austin(공학박사). 1887년~1988년 Univ. of Texas at Austin(Research Fellow). 1996년~1997년 Univ. California-Irvine(Visiting Professor). 관심분야는 이동컴퓨팅, 임베디드 소프트웨어, 유비쿼터스컴퓨팅