MNFS: NAND플래시메모리를 기반으로 하는 모바일 멀티미디어 파일시스템의 설계

(MNFS: Design of Mobile Multimedia File System based on NAND FLASH Memory)

김 효 준 † 원 유 집 †† 김 요 환 †††
(Hyojin Kim) (Youjip Won) (Yohwan Kim)

요 약 본 연구는 NAND 플래시 메모리를 기반으로 하는 모바일 멀티미디어 파일시스템(MNFS)에 관한 내용이다. 이 연구에서 제안하는 파일시스템은 기존 범용 플래시 파일시스템과는 달리 MP3 플레이어, PMP, 디지털 캠코더 등과 같은 모바일 멀티미디어 장치에서 최적의 성능을 보장하기 위하여 설계된 파일시스템이다. MNFS는 크게 세 가지의 특징을 갖는데, 첫째 파일시스템의 순차적인 쓰기 요청에 대해 균일한 응답시간을 보장하고, 둘째 파일시스템 마운트 시간이 빠르며, 셋째 최소한의 메모리만을 소모한다. 이를 위해 4가지 새로운 기법들을 사용한다. 파일시스템 메타데이타와 사용자데이타를 서로 다른 맵핑 기법으로 관리하는 혼합 맵핑 기법과, 파일 데이타 할당 단위를 NAND 플래시 메모리의 블록 단위로 사용한 블록 단위 사용자 데이타 할당, 메타데이타를 최소화 하기 위한 iBAT, 그리고 상향식 디렉토리 표현기법이다.

프로토타입 MNFS를 ARM9 프로세서와 1G 비트 용량의 NAND 플래시 메모리환경에서 구현하고 성능을 측정한다. 기존의 NAND 플래시 파일시스템인 YAFFS와 FTL을 사용하는 FAT파일시스템과 비교하였으며, 순차적 쓰기 요청에 대해 빠르고 균일한 응답시간을 확인할 수 있다. 또, 같은 조건에서 YAFFS에 비해 30배 빠른 마운트 시간을 보였고, Heap 메모리도 YAFFS의 5%밖에 소모하지 않았다.

키워드 : NAND 플래시 메모리, 파일시스템, 모바일, 멀티미디어, 실시간

Abstract Mobile Multimedia File System, MNFS, is a file system which extensively exploits NAND FLASH Memory. Since general Flash file systems does not precisely meet the criteria of mobile devices such as MP3 Player, PMP, Digital Camcorder, MNFS is designed to guarantee the optimal performance of FLASH Memory file system. Among many features MNFS provides, there are three distinguishable characteristics. MNFS guarantees, first, constant response time in sequential write requests of the file system, second, fast file system mounting time, and lastly least memory footprint. MNFS implements four schemes to provide such features. Hybrid mapping scheme to map file system metadata and user data, manipulation of user data allocation to fit allocation unit of file data into allocation unit of NAND FLASH Memory, iBAT (in core only Block Allocation Table) to minimize the metadata, and bottom-up representation of directory.

Prototype implementation of MNFS was tested and measured its performance on ARM9 processor and 1Gbit NAND FLASH Memory environment. Its performance was compared with YAFFS, NAND FLASH File system, and FAT file system which use FTL. This enables to observe constant request time for sequential write request. It shows 30 times faster mounting time to YAFFS, and reduces 95% of HEAP memory consumption compared to YAFFS.

Key words : NAND Flash Memory, File System, Mobile Device, Multimedia, real time

† 비 회 원 : 삼성전자 기술총괄 소프트웨어연구소 선임연구원
zartoven@samsung.com
†† 종신회원 : 한양대학교 전자컴퓨터통신공학과 교수
yjwon@ece.hanyang.ac.kr
††† 비 회 원 : 한양대학교 전자컴퓨터통신공학과
yohwan82@ece.hanyang.ac.kr
논문접수 : 2007년 10월 15일
심사완료 : 2008년 10월 15일

정보과학회논문지: 시스템 및 이론 제35권 제11호(2008.12)

1. 서 론

최근의 모바일 제품의 가장 큰 특징은 멀티미디어 기능의 확장이다. MP3플레이어는 더 이상 벤처기업의 아이디어 상품이 아니라 대기업의 주력 업종이 되었고, 대부분의 필름 카메라가 디지털 카메라로 대치되었다. 또한, PMP(Portable Media Player)를 사용함으로써 고화질의 멀티미디어를 언제 어디서든 감상할 수 있다. 또, 휴대폰에는 카메라 기능과 MP3 플레이어 기능, 그리고 디지털 캠코더 기능까지 장착되고 있으며 심지어 음악을 듣거나 동영상을 감상할 수 있는 전자사전까지 개발되었다.

멀티미디어 기능이 휴대 제품에 채택됨에 따라 휴대용 저장장치의 중요성이 더해졌다. 사용자들은 큰 용량의 멀티미디어 데이타를 휴대하길 원하고, 이를 위해서는 휴대하기에 적합한 저장장치가 필요하다. 휴대 장치에 적합한 저장장치는 휴대제품에 알맞은 크기와, 배터리를 고려한 낮은 전력소모[1]와, 물리적인 충격에 안전해야 한다. 최근 모바일 제품의 멀티미디어화 트랜드의 가장 큰 수혜자는 조건들을 가장 잘 만족 시킬 수 있는 NAND 플래시 메모리며, 그에 따라 관련 기술에 대한 요구사항도 크게 증대하고 있다.

본 논문에서는 모바일 멀티미디어 제품의 요구사항에 맞추어 MNFS (Implementation of Mobile Multimedia File System based on NAND FLASH Memory)를 제안한다. MNFS는 모바일 장치의 스토리지로 널리 사용되고 있는 NAND 플래시 메모리를 기반으로 최적의 성능과 실시간 응답을 제공하면서도 적은 자원을 소모하도록 설계된 파일시스템이다. 기존 멀티미디어 파일시스템이 주로 하드디스크 기반 스트리밍 서버에서 요구되는 다중 프로세스의 읽기 성능 보장을 목표로 한다면, MNFS는 모바일 멀티미디어 기기에서 단일 프로세스의 주기적인 쓰기 요청에 대해 일정한 쓰기 응답시간을 보장하는 것을 목표로 한다. 또 갑작스런 전원차단에 대한 안정성을 제공하며, 최소의 코드와 데이타만을 사용하도록 설계한다.

본 논문의 구성은 다음과 같다. 3장에서는 MNFS의 설계에 대해 설명하고, 4장에서는 MNFS, YAFFS, FAT파일시스템의 비교실험을 통해 성능을 보이고, 마지막 5장에서는 결론을 맺는다.

2. 관련 연구

기존 멀티미디어 파일시스템 연구의 대상은 멀티미디어 데이타를 스트리밍 하는 서버였다. 하드디스크는 자기식으로 데이타를 읽고 쓰기 때문에 읽기속도와 쓰기속도가 같다. 읽기 작업과 쓰기 작업의 비용이 같을 때, 병목현상이 발생할 곳은 읽기 작업이다. 여러 개의 멀티미디어 데이타를 동시에 저장하는 일은 흔하지 않지만, 여러 개의 멀티미디어 데이타를 동시에 읽는 일은 빈번히 발생하기 때문이다. 즉, 기존 하드디스크 기반의 멀티미디어 파일시스템은 여러 개의 프로세스가 요청하는 멀티미디어 파일의 읽기 요청들을 어떻게 효율성 있게 처리할 것인지가 목표이다.

플래시 메모리의 특성은 하드디스크와 다르다. 쓰기 작업일 경우는 읽기 작업보다 복잡하고 시간도 오래 걸린다. 시간적으로 플래시 메모리의 쓰기 작업은 하드디스크에 비해서 느린 반면 플래시 메모리의 읽기 속도는 하드디스크보다 더 빠르거나 같다. 이러한 저장 미디어의 특성 차이는 기존 하드디스크를 기반의 멀티미디어 파일시스템과 플래시 메모리를 기반의 모바일 멀티미디어 파일시스템간의 중요한 차이점이다. 또 모바일 멀티미디어 기기에서 여러 개 프로세스에 의한 다중 입출력 요구가 거의 일어나지 않는다는 점 또한 기존 연구와의 차이점이다. 이러한 특성 때문에 플래시 메모리를 기반으로 하는 모바일 멀티미디어 파일시스템에서는 단일프로세스의 단일 스트리밍에 대한 쓰기 요청을 어떻게 안정적으로 처리할 것인가가 가장 중요한 연구 목적이 된다.

모바일 멀티미디어 제품들에 가장 널리 사용되고 있는 저장장치는 NAND 플래시 메모리이다. NAND 플래시 메모리는 전력 소모가 작고 충격에 강하며 소형화가 가능한 장점이 있는 반면에, 자기 저장장치인 하드 디스크와 달리 섹터단위 갱신이 불가능한 단점이 있다[2-4]. 즉, 쓰기 작업 이전에 지우기 작업이 선행 되어 있어야 한다는 것과, 읽기/쓰기 작업의 단위보다 지우기 작업의 단위가 훨씬 크다는 것이다. NAND 플래시 메모리의 경우 읽기/쓰기 작업은 512바이트 혹은 2048바이트 크기를 가지는 페이지 단위로 이뤄지고, 지우기 작업은 이러한 페이지들이 여러 개 모인 블록 단위로 이뤄진다. NAND 플래시 메모리는 용량에 따라 여러 개의 블록으로 구성이 된다. 하나의 블록은 32개의 페이지로 이뤄져 있으며, 하나의 페이지는 다시 내부적으로 512바이트의 메인 영역과 16바이트의 스페어 영역으로 구성된다[2]. NAND 플래시 메모리에서 읽기/쓰기 작업은 페이지 단위로 이뤄지며, 하나의 페이지에 대해서 메인 영역과 스페어 영역에 대해서 각각 읽고 쓰거나 혹은 한번에 읽고 쓸 수 있다. 메인 영역은 사용자 데이타를 저장하기 위한 공간이며, 스페어 영역은 특별한 목적으로 사용하는 예비 공간이다[2]. 스페어 영역은 대표적으로 매인 영역에 대한 오류정정코드(ECC)를 보관하거나 베드블록마크를 보관한다. NAND 플래시 메모리의 용량을 말할 때에는 스페어 영역은 포함시키지 않는다. 예를 들어 32개의 페이지로 구성된 블록 하나의 용량은 16 Kbytes

이고, 128MBytes용량인 NAND 플래시 메모리에는 총 8,192개의 블록이 존재한다.

한편, 현재 모바일 제품에 가장 널리 사용되고 있는 FAT파일시스템은[5] 하드디스크나 플로피 디스크와 같은 자기저장장치를 위해 설계된 파일시스템으로서 앞서 언급한 독특한 특성을 갖는 NAND 플래시 메모리에는 직접 사용할 수 없다. 이런 문제를 해결하기 위해 사용하는 소프트웨어가 FTL(Flash Translation Layer)이다[6]. FTL은 파일시스템과 플래시메모리의 중간에 위치하여 플래시메모리가 마치 하드디스크처럼 보이도록 하는 역할을 한다. 즉, 논리섹터에 대한 쓰기 요청을 쓰기 가능한 물리섹터에 처리하고 그 맵핑 정보를 따로 관리하는 방법으로[7], 플래시 메모리가 마치 섹터단위의 갱신이 가능한 것처럼 보이도록 한다. 그렇기 때문에 FTL을 사용하면 FAT파일시스템과 같은 기존 하드디스크용 파일시스템을 플래시 메모리상에서 사용할 수 있게 된다.

하지만 FAT파일시스템과 FTL을 사용하는 시스템에는 몇 가지 문제가 있다. 첫째, 파일시스템과 FTL의 분리된 레이어를 통해 작업이 이뤄지기 때문에 최상의 성능을 얻기 힘든 문제가 있다. NAND 플래시 메모리의 성능향상이 이뤄지고 있지만 모바일 제품에 사용되는 멀티미디어 데이타의 정보량도 또한 급속히 증가하고 있다는 것을 감안하면 보다 최적의 성능을 얻기 위하여 개선이 필요하다. 둘째, FAT파일시스템은 안정성이 부족하다. 모바일 제품은 그 특성상 갑작스런 전원 차단의 문제가 발생할 수 있다. FAT파일시스템은 동작 도중 전원이 차단되었을 경우, 데이타가 소실되거나 파일시스템 전체가 깨질 수 있기 때문에 중요한 사용자 정보의 유실이나 제품의 고장을 초래할 위험이 있다. 셋째, FTL과 FAT파일시스템이 모두 실시간 응답을 보장하지 못한다. 특히, FTL은 같은 쓰기 작업에 대해서도 그 응답시간이 일정하지 않다. 이것은 FTL 내부에서 갱신이 어려운 플래시메모리의 문제를 해결하기 위해 내부 알고리즘에 의한 블록 삭제작업과 페이지 복사작업 등을 처리하기 때문이다. 그러므로 파일시스템 유저는 같은 크기의 쓰기 요청이 언제 끝날지를 예측할 수 없으며, 이는 시스템 설계에 어려움을 야기하기도 한다.

이러한 문제들을 근본적으로 극복하기 위해 플래시 메모리를 위한 전용 파일시스템을 설계하고자 하는 노력이 있어 왔다. 그 중 가장 성공적인 파일시스템으로 JFFS2(Journaling Flash File System 2)[8]와 YAFFS (Yet Another Flash File System)[9]가 있는데, JFFS2는 1999년 발표된 JFFS를 발전시킨 것으로, LFS(Log-Structured File System) 구조를 사용한다[10]. LFS는 파일시스템 메타데이타와 일반데이타를 묶어 로그라는 자료구조를 만들고, 이 로그를 큐 형태의 디스크에 계속 추가시켜 나가는 방식으로 동작한다.

플래시 메모리를 위한 소프트웨어는 크게 3가지로 분류할 수 있다. MTD(Memory Technology Devices), FTL(Flash Translation Layer), Flash File System이다. 먼저 MTD는 Linux에서 플래시 메모리를 위해 정의한 드라이버 모델이다. 플래시 메모리를 위한 일종의 디바이스 드라이버라고 보아도 무관하며, 플래시 메모리의 읽기, 쓰기, 지우기 등을 담당하는 API를 제공한다. FTL은 플래시 메모리를 업데이트가 가능한 블록디바이스처럼 보이도록 하는 소프트웨어다. MTD가 Linux에서만 사용되고 있는데 반해, FTL은 Linux외의 여러 환경에서도 사용되고 있다. 마지막으로 Flash File System은 플래시 메모리를 위해 새롭게 설계된 파일시스템을 말한다. JFFS, JFFS2, YAFFS, 등이 이에 해당하며 Linux환경에서는 FTL을 없이 MTD를 직접 사용하는 것이 특징이다.

3. MNFS 설계

MNFS 설계의 핵심 기법은 다음 네 가지로 요약할 수 있다. 핵심기법으로는 (1) 혼합 맵핑 기법 (2) 블록단위의 사용자 데이타 할당 기법 (3) iBAT (4) 상향식 디렉토리 표현 기법이다. 먼저 3.1장에서 3.4장까지 MNFS의 설계의 4가지 핵심에 대해 설명하고, 마지막으로 3.5장에서 순차적인 쓰기 요청에 대한 응답시간이 어떻게 예측되는지를 설명한다.

3.1 혼합 맵핑 기법

플래시 메모리를 대상으로 FTL이나 파일시스템을 설계할 경우 가장 먼저 해결해야 할 문제는 하드디스크와 다른 플래시 메모리의 특성을 어떻게 해결할 것인지를 결정하는 것이다. 플래시 메모리는 페이지 단위로 읽기 쓰기가 가능하지만 업데이트를 위해서는 블록단위의 지우기 작업을 먼저 수행해야 한다. 블록은 여러 페이지의 집합으로 구성되어 있기 때문에, 갱신하고자 하는 페이지 이외에 다른 페이지의 내용까지 지워지는 문제가 있다. 이러한 특성을 극복하기 위하여 몇 가지 방법이 사용되고 있다. 대표적으로 블록 맵핑 방법과 페이지 맵핑 방법이 있다. 블록 맵핑 방법은 스마트카드 표준에서 사용하는 가장 간단한 방법 중의 하나이며, 블록 맵핑 방법에서 페이지의 내용을 갱신하기 위한 절차는 다음과 같다. 새로운 블록을 할당하고, 원래 블록으로부터 갱신할 페이지만 제외하고 나머지 페이지들을 새로운 블록으로 복사한 다음, 갱신할 페이지를 새로운 블록에 기록한다. 그리고 마지막으로 새로운 블록이 원래 블록을 대체하도록 블록단위의 맵핑 테이블을 수정하면 페이지 갱신이 종료된다. 그림 1은 블록 맵핑 기법의 동작 과정이다. 블록당 페이지 수를 4개로 가정하고, 그 중 2번

섹터가 갱신된다고 가정한다. 블록 맵핑을 사용할 때에는 블록 단위의 맵핑 테이블이 메모리상에 존재하게 된다. 이 맵핑 테이블을 사용하여 논리적인 블록 번호를 통해 물리적인 블록 번호를 찾을 수 있다. 그림 1과 같이 초기에 논리블록 0번이 물리블록 0번으로 맵핑 되어 있음을 알 수 있다. 임시블록은 물리블록 100번이 사용되고 있는데, 2번 섹터를 갱신하는 과정에서 결국 논리블록 0번이 물리블록 100번으로 바뀌었음을 알 수 있다. 또, 기존 0번 물리블록은 임시블록으로 바뀌었다.

페이지 맵핑 방법은 YAFFS등에서 사용되는데, 페이지의 내용을 갱신하기 위한 절차는 블록 맵핑에 비해 훨씬 간단하다. 갱신하고자 하는 논리 페이지 내용을 임의의 비어있는 물리 페이지에 기록하고, 해당 논리 페이지에 대한 페이지 단위의 맵핑 정보를 기록된 물리 페이지를 가리키도록 변경하면 된다.

그림 2는 페이지 맵핑 방법의 동작 과정이다.

페이지 맵핑 방식은 페이지 단위의 큰 맵핑 테이블을 사용하는 대신 업데이트에 따른 오버헤드가 매우 작다. 반면 블록 맵핑 방식은 블록 단위의 맵핑 테이블만 사용하기 때문에 적은 메모리를 소모하지만, 페이지 단위

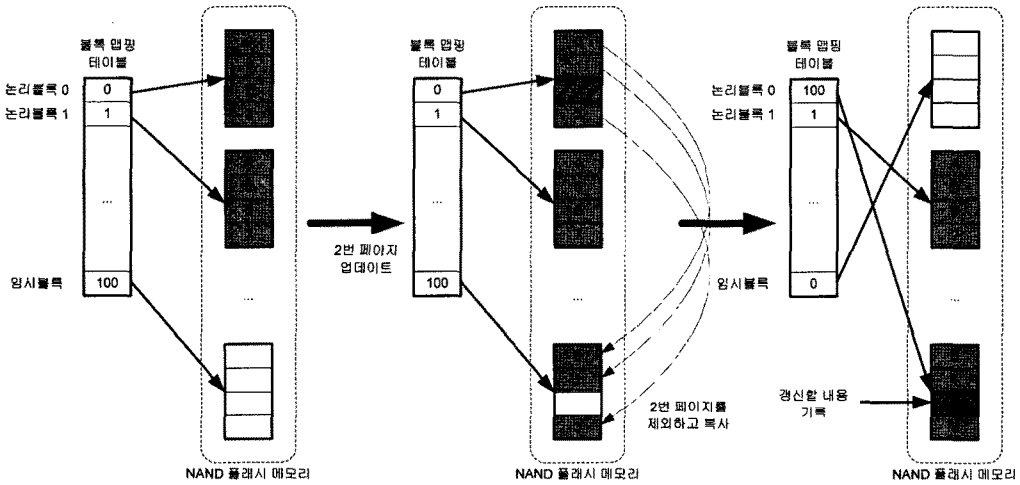

그림 1 블록 맵핑 기법의 동작

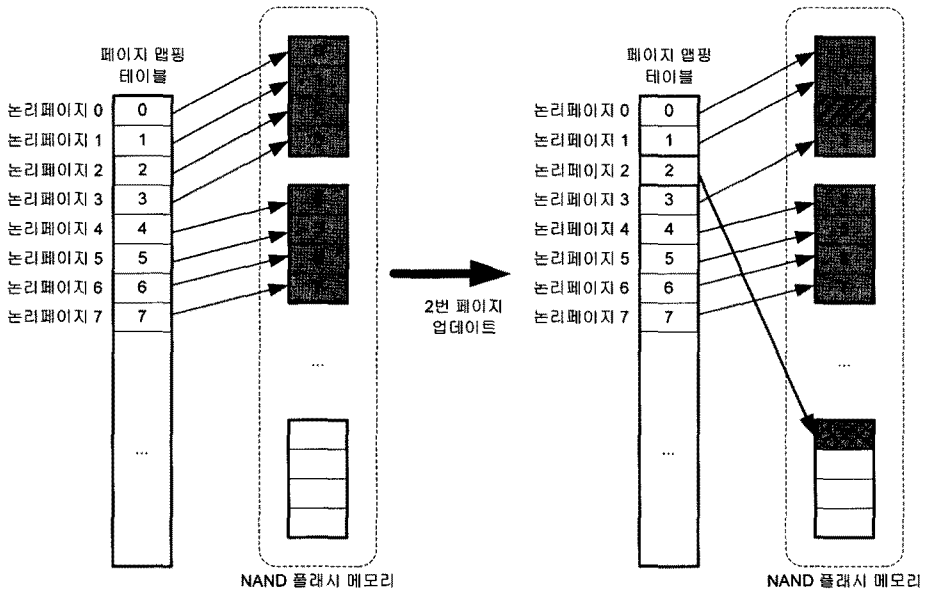

그림 2 페이지 맵핑 기법의 동작

표 1 페이지 맵핑과 블록 맵핑의 비교

	블록 맵핑	페이지 맵핑
섹터 갱신작업의 효율성	비효율적	효율적
맵핑 테이블의 크기	작다	크다

의 갱신이 일어날 경우엔 큰 오버헤드가 발생하는 단점이 있다. 페이지 맵핑과 블록 맵핑에 대한 비교는 표 1과 같다.

MNFS에서는 관리하는 데이타의 종류에 따라 페이지 맵핑 방법과 블록 맵핑 방법을 혼용하는 혼합 맵핑 기법을 사용한다. MNFS의 혼합 맵핑 기법은 업데이트가 잦은 파일시스템 메타데이타는 페이지 맵핑 기법을 사용하여 관리하고 업데이트가 자주 일어나지 않는 사용자 데이타는 블록 맵핑 기법을 사용하여 관리한다. 이러한 혼합 맵핑 기법을 사용함에 따라 메모리 사용량을 최소화시키며, 파일시스템 성능을 향상을 갖는다.

그림 3은 MNFS의 파일시스템 기본 구조로써 메타데이타 영역과 사용자데이타 영역이 구분되었지만, 이것은 이해를 돕기 위한 개념적인 것으로 실제로는 블록단위로 NAND 플래시 메모리상에 영역 구분 없이 기록되며, 로그블록에는 MNFS의 메타데이타가, 데이타블록에는 사용자 데이타가 기록된다.

MNFS의 모든 로그블록은 할당된 순서에 따라 리스트를 구성하며, 순차적으로 증가하는 로그블록번호를 사용하여 그 순서를 유지한다. MNFS의 로그블록 첫째 페이지에는 메인 영역과 스페어영역에 대한 각각의 로그블록헤더와 로그블록번호가 기록되고, 이외의 페이지는 MNFS의 메타데이타인 디렉토리 엔트리를 기록하는 목적으로 사용한다. 디렉토리 엔트리는 각각 하나의 파일이나 폴더에 대응되며, 파일과 디렉토리에 대해서 각각의 이름, 생성날짜, 크기 등의 정보를 가지며, 디렉토리는 부모 디렉토리 엔트리의 번호도 포함된다. 디렉토리 엔트리는 각각의 고유 아이디를 가지며, 각각의 고유 아이디는 페이지의 스페어 영역에 기록된다. 디렉토리 엔트리는 로그블록에 시간에 따라 순차적으로 기록되며, 같은 아이디를 가지는 여러 개의 디렉토리 엔트리가 존재할 경우, 가장 나중에 기록된 엔트리가 유효하다. 그림 4는 로그블록 구조이다.

NAND 플래시 메모리의 페이지 크기를 고려하여 512 바이트 크기를 갖도록 설계한다.

3.2 블록 단위의 사용자 데이타 할당 기법

기존 파일시스템에서는 파일에 데이타를 할당하기 위해 4Kbytes 정도의 크기를 가지는 클러스터 단위를 사용한다. 하지만 멀티미디어 파일시스템에서는 비교적 큰 크기의 멀티미디어 파일 크기를 고려하여 상대적으로 큰 크기의 클러스터를 사용하는 것이 일반적이다. MNFS 역시 비교적 큰 NAND 플래시 메모리의 블록을 할당단위로 사용한다. 이것은 MNFS는 아무리 작은 파일이라

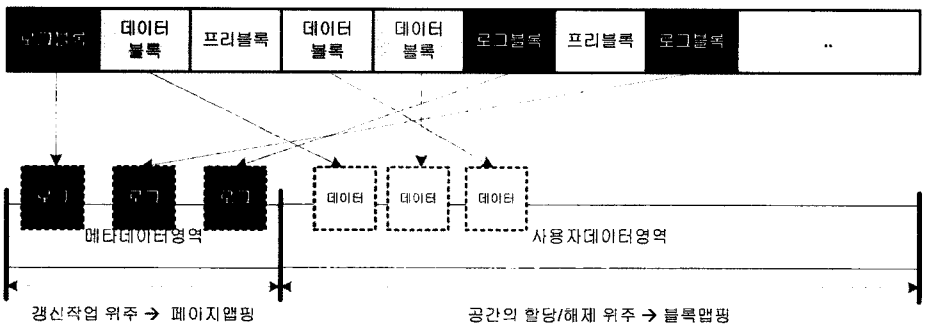

그림 3 MNFS 파일시스템의 기본 구조

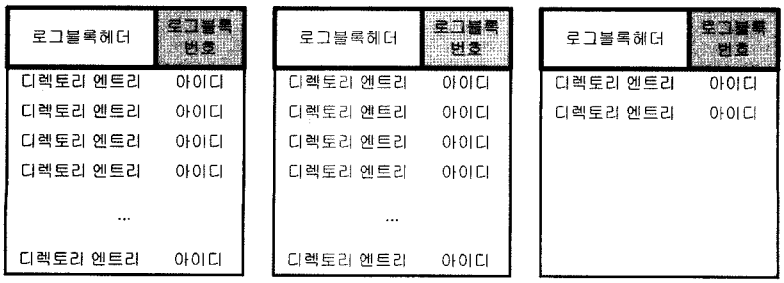

그림 4 로그블록의 구조

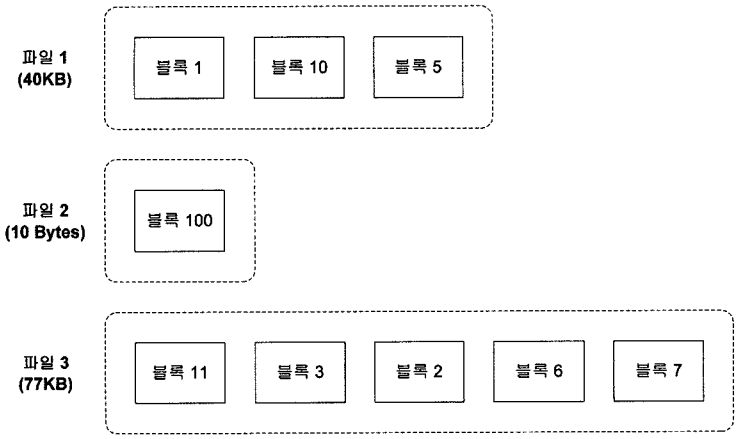

그림 5 블록 단위의 사용자 데이타 할당

도, 하나의 블록을 할당 한다는 것이다. 그림 5는 블록 단위의 사용자 데이타 할당에 대한 그림이다.

NAND 플래시 메모리에서는 지우기 단위가 블록이다. MNFS에서는 사용자 데이타 할당을 블록 단위로 하기 때문에 파일을 삭제할 때 해당 파일의 데이타를 저장하던 모든 블록을 지운다. 파일을 지울 때 NAND 플래시 메모리의 블록을 지워둘 수 있다는 것은 대단히 중요한 의미를 지닌다. NAND 플래시 메모리를 저장장치로 사용하는 과정에는 결국 페이지 단위의 쓰기 작업들과 블록 단위의 지우기 작업들이 포함된다. NAND 플래시 메모리의 블록 삭제 작업은 페이지 쓰기 작업보다 약 10배 정도 더 오래 걸리는 작업이다. 이렇게 오래 걸리는 지우기 작업을 실제 파일이 지워질 때 할 수 있다는 것은 중요한 의미를 갖는다. MNFS에서는 파일을 지울 때 정보를 갖고 있지 않는 블록들을 미리 지워 놓기 때문에 이후 쓰기 작업에서는 항상 이미 지워져 있는 블록을 할당 받을 수 있다. 즉, 시간이 오래 걸리는 지우기 작업 없이 쓰기작업을 처리할 수 있어, 균등한 쓰기응답이 가능해 진다. 기존의 플래시 파일시스템인 YAFFS나 JFFS2 등에서는 파일을 삭제할 때 블록을 지워 둘 수 없다. 따라서 쓰기 작업을 진행하다 공간이 부족하면 가비지컬렉션을 수행하고[11], 그 과정을 통해 블록들을 지운다. 따라서 쓰기 작업의 응답이 극단적으로 느려진다.

MNFS에서 파일을 삭제하는 작업은 기존 파일시스템에 비해 더 오래 걸린다. 이 부분은 현재 버전의 단점이며, 앞으로 개선이 가능하다. 최신의 NAND 플래시 메모리에서 제공되는 Multiple Block Erase기능을 사용할 수 있는데, 기존 블록 삭제 작업이 2ms가 걸리는데 반해, 이 기능을 사용하면 최대 64개의 블록을 4ms 동안에 지울 수 있다. 현재 버전에서는 이 기능이 구현되지 않았지만, 이 기능을 구현할 경우 파일 삭제작업의 시간도 기존에 비해 크기 느리지 않으면서 최적의 쓰기 속도를 보장할 수 있다. 또, Multiple Block Erase가 제공되지 않는 NAND플래시 메모리에서는 백그라운드 태스크를 두어 시스템이 한가할 때 블록 삭제 작업을 처리하도록 하는 방법도 있다.

블록단위 데이타 할당의 또 다른 장점은 파일시스템에 필요한 메모리를 최소화시킬 수 있다는 점이다. 다음에 설명하는 iBAT은 블록 단위 사용자 데이타 할당을 기반으로 저장공간의 상태를 관리하는 MNFS의 자료구조다.

3.3 iBAT: 호스트에만 존재하는 블록 할당 테이블

MNFS에서는 FAT파일시스템의 FAT테이블과 유사한 iBAT을 사용한다. iBAT은 FAT과 유사하지만 두 가지 면에서 차이를 갖는다. 첫째는 iBAT의 경우 저장장치 안에 존재하지 않고 파일시스템을 마운트 할 때 블록 스캔을 통해 동적으로 구축된다. 또 다른 차이점은 할당 단위로써, FAT의 경우 클러스터 단위로 할당을 하는데 반해 iBAT는 NAND 플래시 메모리의 블록 단위로 할당을 하는 점이 다르다. 이러한 설계의 목적은 균일한 쓰기 응답시간을 보장하기 위함과 파일시스템의 안정성을 높이기 위함이다. FAT파일시스템의 경우 파일의 크기가 증가함에 따라 주기적으로 새로운 클러스터를 해당 파일에 할당해야 하고, 이 과정에서 디스크의 특정 영역에 존재하는 FAT테이블을 업데이트해야 한다. FAT테이블을 업데이트 하는 일 자체가 파일시스템의 쓰기 작업의 응답시간에 영향을 미치게 되고, 이 작업 도중에 전원이 차단 됐을 경우엔 FAT 테이블의 내용에 문제가 생기는 등, 안정성 측면에서도 악영향을 끼친다.

MNFS의 iBAT의 경우, 사용자 데이타가 저장되는

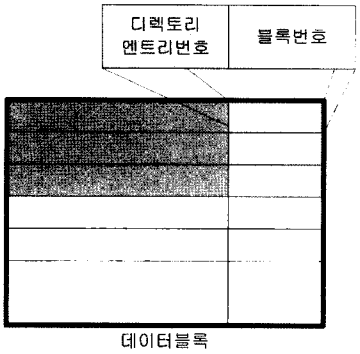

그림 6 MNFS의 데이타블록

데이타블록의 스페어 영역의 스캔 과정을 통해 파일시스템이 마운트 될 때 재구축된다. 즉, 파일의 크기가 커짐에 따라 별도의 파일시스템 메타데이타의 수정이 필요 없이, 쓰려고 하는 블록의 스페어 영역에 파일 ID와 블록 번호를 기록하는 방식으로 블록 할당 작업이 처리된다. 이 작업은 NAND 플래시 메모리의 메인 영역에 사용자 데이타를 기록함과 동시에 처리될 수 있어 별도의 시간을 소모하지 않기 때문에, 항상 일정한 쓰기 응답시간을 보장할 수 있다. 또, FAT테이블과 같은 파일시스템 메타데이타가 디스크 상에 아예 존재하지 않으므로 해당 데이타 때문에 파일시스템이 망가지는 일이 없게 된다. 데이타블록의 구조는 그림 6과 같다. 데이타블록의 스페어 영역에 기록되는 디렉토리 엔트리 번호는 블록이 포함된 파일을 가리키는 역할을 하고, 블록 번호는 해당 파일 내에서 논리적으로 몇 번째 위치하는 블록인지를 나타낸다.

MNFS에서 파일시스템을 최초 마운트하며, iBAT을 구축하는 순서는 다음과 같다. 먼저 NAND 플래시 메모리의 모든 블록에 대해 첫째 페이지의 스페어 영역을 읽는다. 이 과정에서 해당 블록의 타입을 판별한다. MNFS에서는 4종류의 블록이 사용되는데, 디렉토리 엔트리, 즉 메타데이타를 저장하기 위한 로그블록과 사용자 데이타를 저장하기 위해 사용되는 데이타블록, 지워진 상태로 데이타블록이나 로그블록으로 사용될 수 있는 프리블록, 그리고 하드웨어적인 결함으로 사용해서는 안 되는 베드블록이다. 블록 스캔작업을 통해 로그블록들을 골라낸 다음, 각 로그블록들의 ID에 따라 생성된 순서대로 리스트를 구성한다. 리스트가 구성되면 각 로그블록의 스페어영역을 순서대로 따라가며 읽는다. 스페어 영역에 기록된 디렉토리 엔트리 번호를 통해 유효한 디렉토리 엔트리를 찾는다. 이 과정이 끝나고 나면, 유효하다고 선별된 디렉토리 엔트리들을 읽어 들이고 해당 디렉토리 엔트리가 파일을 가리키는 디렉토리 엔트리일 경우, 해당 파일에 속하는 데이타블록들을 선별하여 iBAT으로 구성한다. 이때 맨 처음 스페어 영역 스캔 작업에서 읽어 두었던 정보를 사용하기 때문에 추가적인 읽기 작업이 필요하진 않다. 이상의 과정을 유효한 모든 디렉토리 엔트리를 대상으로 반복하여 iBAT 구축 작업을 완료한다.

3.4 상향식 디렉토리 표현 기법

MNFS는 기존 파일시스템에서 사용하던 디렉토리 파일을 사용하지 않도록 설계되었다. 디렉토리 파일은 디렉토리에 포함된 파일 또는 서브 디렉토리의 이름과 정보를 가진 파일로 일반 데이타 파일과 똑같은 방식으로 유지 관리된다. MNFS에서 이 방법을 사용하지 않는 이유는, 이 파일의 경우 MNFS에서 가정한 멀티미디어 파일과는 달리 빈번한 갱신작업이 일어날 수 있기 때문이다. MNFS에서는 멀티미디어 파일의 경우 갱신작업보다는 순차적인 쓰기작업 위주로 일어난다고 가정한다. 본 논문에서는 MNFS에서 디렉토리 파일 대신 사용할 수 있는 방법으로, 상향식 디렉토리 표현기법을 제안하였다. 이 기법에서는 각각의 디렉토리 엔트리가 자신이 포함된 부모 디렉토리 엔트리 번호를 포함한다. 디렉토리에 포함된 자식 디렉토리 엔트리가 부모 디렉토리를 가리키기 때문에 상향식이라고 부른다.

예를 들어 표 2와 같이 디렉토리 엔트리 번호가 구성되어 있을 경우, 디렉토리 구조는 그림 7과 같이 표현된다.

이러한 방법을 사용하려면 항상 모든 디렉토리 엔트리를 읽어서 계층적 구조에 대한 정보를 메모리상에 구축해 놓아야 한다. 특히 파일의 개수가 많을 경우 심각한 성능저하와 자원소모가 발생할 수 있는 위험이 있다.

표 2 상향식 디렉토리 표현방식의 예

디렉토리 엔트리 번호	이름	부모 디렉토리 엔트리 번호
0	MNFS Volume	0
1	Animals	0
2	Birds	1
3	Tiger	1
4	Eagle	2

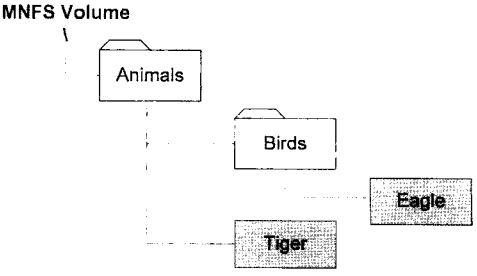

그림 7 상향식 디렉토리 표현

다만, MNFS에서는 대상으로 하는 파일의 크기가 NAND 플래시 메모리의 용량에 비해 상대적으로 커서, 파일의 개수 또한 제한적일 수 밖에 없기 때문에 그러한 문제가 심각하지 않다. 예를 들어, 1Gbytes의 용량을 가지는 MP3플레이어를 가정할 경우, 4Mbytes의 MP3파일이 256곡 저장될 수 있다. 이 경우 상향식 디렉토리 표현기법을 사용하면 256회의 페이지 읽기 작업과 512Bytes의 메모리밖에는 소모되지 않는다.

3.5 쓰기 응답시간의 보장

MNFS는 (S_{main} * N) 바이트의 순차적 쓰기 요청을 N번의 페이지 기록으로 처리하도록 설계되었다. 여기서 S_{main}은 NAND 플래시의 한 페이지의 메인 영역의 크기로, 소블록 플래시 메모리에서는 512이고 대블록 플래시 메모리에서는 2,048이다. 예를 들어 소블록 플래시 메모리에서 32Kbytes의 쓰기 요청은 64회의 페이지 쓰기로 처리된다. 물론 CPU연산 오버헤드가 조금 붙긴 하지만 이 시간은 페이지 기록 시간에 비해 상대적으로 매우 짧다.

NAND 플래시 메모리의 한 페이지에 대한 쓰기시간은 플래시 메모리로 쓰고자 하는 데이타를 전송하는 시간과, 전송된 데이타를 실제로 기록하는 시간으로 이뤄진다. 데이타 전송시간은 플래시 메모리와 CPU의 연결 버스 동작 속도에 의존적이다. 플래시 메모리의 페이지 기록 시간은 플래시 디바이스에 따라 달라지는데, 보통 200us 정도의 시간이 걸린다.

NAND 플래시 메모리에서 한 페이지를 기록하는 시간을 t_{page}라고 할 때, t_{page}는 다음과 같이 정의될 수 있다.

$$t_{page} = (S_{page} \times t_{tr}) + t_{prog}$$

이때 S_{page}는 페이지의 크기로, 소블록 플래시메모리에서는 528, 대블록 플래시메모리에서는 2,112이다. t_{tr}은 바이트당 전송시간이고, t_{prog}는 전송된 페이지를 기록하는 시간이다. 따라서 (S_{main} * N) 바이트의 쓰기 요청에 대해 CPU오버헤드를 무시한 MNFS의 응답시간은 다음 식으로 정의될 수 있다.

$$((S_{page} \times t_{tr}) + t_{prog}) \times N$$

4. 성능평가

4.1 실험 환경

MNFS의 성능평가 및 기존파일시스템과의 비교를 위해, MNFS와 YAFFS, FAT파일시스템을 비교한다. 203Mhz로 동작하는 ARM920T 프로세서를 사용하는 SMDK2410 보드를 사용하고, 1Gbits용량의 소블록 NAND 플래시 메모리를 사용한다. FAT파일시스템은 pSOS2.5에 포함된 pHILE을 사용한다.

4.2 실험 및 결과

실험은 모바일 멀티미디어 파일시스템이 사용되는 환경을 감안하여 총 5가지의 시나리오로 실험을 진행한다. 각 시나리오에 대해 YAFFS, FAT, MNFS 3종의 파일시스템 성능을 비교한다.

시나리오 1: 123MB공간을 1~5 MB 크기의 파일로 채우는 실험

128MB공간을 FAT파일시스템을 사용하여 포맷할 경우, FTL오버헤드와 FAT파일시스템 오버헤드 등을 제외하고 총 123MB의 공간이 남는다. FAT 파일 시스템의 남은 공간에서 1~5 Mbytes까지 1Mbytes 단위의 임의의 크기의 파일을 생성하며 각 파일이 생성되는 시간을 측정한다. 그림 8의 결과를 보면, YAFFS가 평균 686KB/s의 쓰기 속도를, FAT은 278KB/s, 그리고 MNFS는 1,538KB/s의 쓰기속도를 나타낸다. 각 속도에 대한 분산 값은 각각 18, 631, 49이다. YAFFS는 알고리즘상 가비지컬렉션 작업을 하지 않을 경우 이론적으로 MNFS와 같은 결과를 가져야 하지만, 실제로는 절반 이하의 속도를 보인다. 이것은 YAFFS가 MNFS에 비해 상대적으로 복잡한 알고리즘을 사용하고 있어 프로세서의 연산 오버헤드가 크기 때문이다. 이를 확인하기 위해 시나리오1의 실험과정에서의 쓰기 회수를 측정한 결과, MNFS와 YAFFS의 NAND쓰기회수가 거의 비슷하다. 따라서 복잡한 자료구조로 인한 CPU연산 오버헤드 때문임을 알 수 있다. YAFFS는 쓰기 작업 도중에 자료구조를 동적으로 생성하고 삭제한다. 특히 이 과정에 사용되는 `malloc()/free()` 함수의 오버헤드가 매우 크다.

시나리오 2: 시나리오 1을 마친 상태에서 빈 공간 64MB이상을 확보할 때까지 임의의 파일을 선택하여 삭제한다.

파일단위의 삭제 작업을 통해 파일삭제에 소모되는 시간을 비교한다. 시나리오 1이 끝난 상태에서 다시 임의의 파일을 선택하여 삭제하여, 디스크에 조각이 발생하도록 한다. 그림 9의 결과를 보면, YAFFS와 FAT파일시스템에 비해 MNFS의 파일삭제 시간이 오래 걸리는 것을 알 수 있다. 그래프에서 X축은 삭제하는 파일의 크기이고 Y축 시간은 파일 삭제에 소요된 시간을 파일 크기로 나눈 값이다. 즉, 1Mbytes당 삭제시간이다. MNFS의 경우 YAFFS나 FAT파일시스템에 비해 파일 삭제 작업이 오래 걸리는데, 이것은 MNFS가 파일삭제 작업에서 할당되었던 블록을 모두 지워놓기 때문이다. 반면 YAFFS와 FAT에서는 메타정보의 수정만으로 파일삭제 작업을 마친다. 따라서 해당 공간에 대한 가비지 컬렉션이 이후 쓰기 작업 도중 발생하게 된다.

시나리오 3: 시나리오2를 마친 상태에서 다시 남은

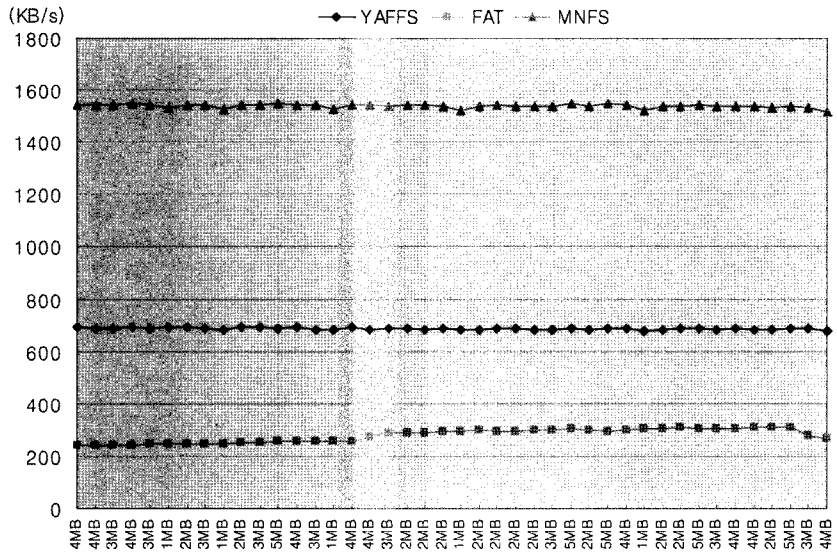

그림 8 시나리오 1의 실험 결과

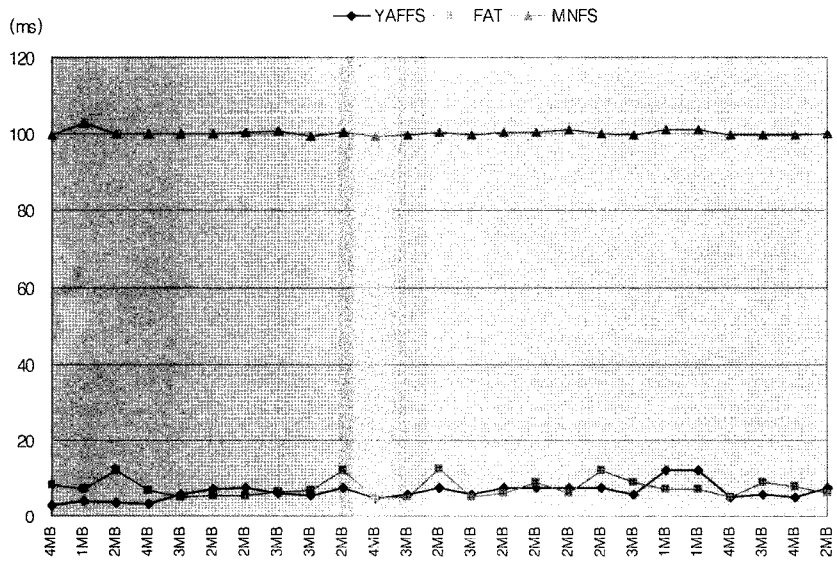

그림 9 시나리오 2의 실험결과

공간을 1~5Mbytes 파일로 채우는 실험

이번 시나리오에서는 시나리오1과 시나리오2를 통해 조각나있는 볼륨에 대해, 다시 파일을 생성하며 쓰기 성능을 측정한다. 시나리오1과 같은 방법을 사용하여 64M bytes공간을 채운다. 그림 10의 결과를 보면 YAFFS의 경우 처음 2MB파일을 생성할 때, 가비지컬렉션 작업이 일어났음을 예측할 수 있다. 가비지컬렉션 작업을 통해 가용한 공간을 확보한 다음에는, 다시 650KB/s 정도의 쓰기성능을 유지한다. FAT파일시스템의 경우엔, 그 응답이 매우 불규칙하며, 이것은 사용하는 FTL의 알고리즘에 의한 것으로, 파일시스템에서 파일을 지우더라도 그 사실을 FTL에서는 알지 못하고, 시나리오 1의 작업을 통해 전체 볼륨이 가득 차게 되고 이로서 FTL의 성능에 영향을 주게 된다. 이에 비해 MNFS는 약 1,500KB/s의 성능이 유지된다.

시나리오 4: 32KB단위로 64MB파일 쓰기 응답시간 측정

시나리오 3을 마친 상태에서, 시나리오 2의 방법으로

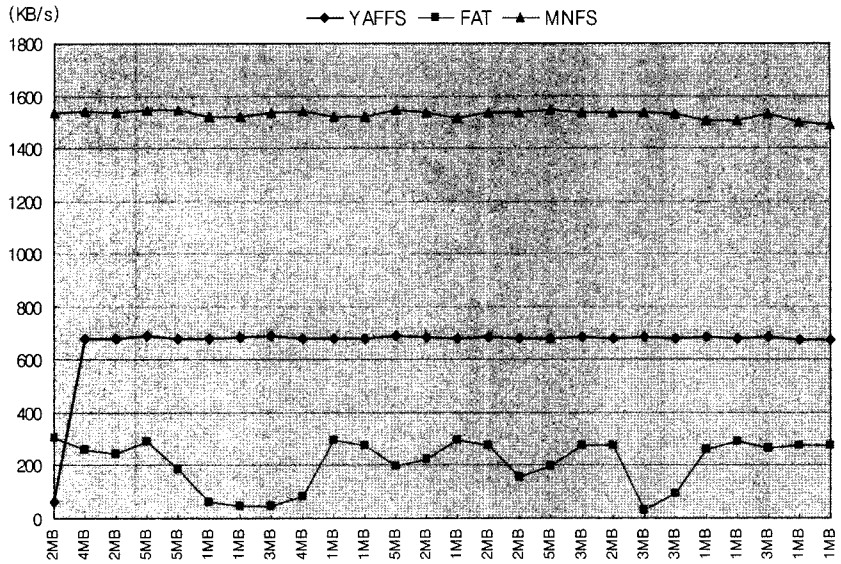

그림 10 시나리오 3의 실험결과

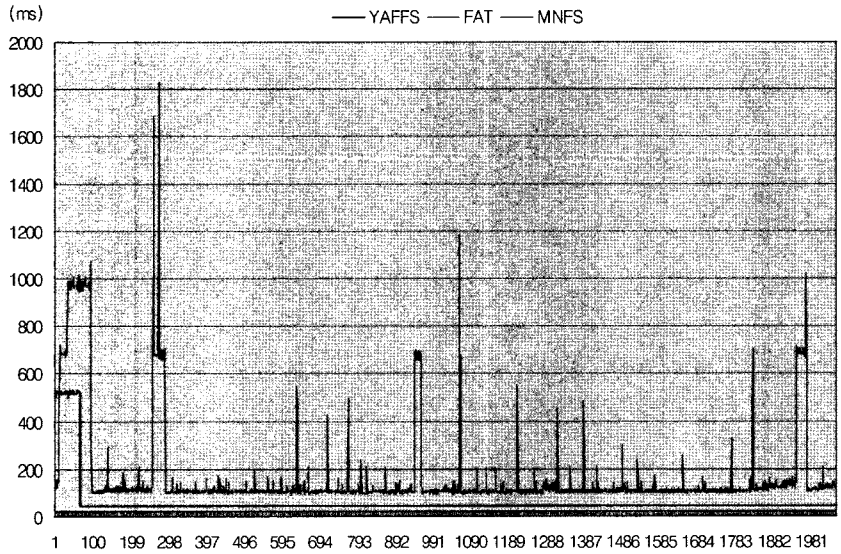

그림 11 시나리오 4의 실험결과

볼륨에 64MB의 공간을 확보한 다음, 하나의 파일을 만들고, 32KB씩 2048회 쓰기 요청을 하여 각 쓰기 요청에 대한 응답시간을 측정하는 실험이다. 이 실험을 통해 멀티미디어데이타의 녹음/녹화 작업에 대해 파일시스템이 얼마나 균등한 쓰기 응답시간을 보이는지 비교할 수 있다. 그림 11을 보면 YAFFS는 초기 약 520ms 정도의 응답시간을 보인 후 약 50ms의 응답시간을 꾸준히 나타냄을 알 수 있다. 이것은 앞 부분에서 YAFFS의 가비지컬렉션 작업이 일어났다는 것을 의미한다. 볼륨이 가득 찬 상태에서 FTL의 성능 저하 때문에, FAT파일시스템의 경우 응답이 매우 불규칙하다. 이에 반해, MNFS의 경우 평균 21ms, 분산 0의 결과를 보인다. YAFFS의 평균 응답시간 62ms과 분산 값 7,097을 나타내며, FAT파일시스템은 174ms의 평균 응답시간과 41,248의 분산 값을 갖는다.

이상의 실험을 통해 MNFS가 연속된 파일쓰기 요청에 대해 균등한 쓰기 응답을 보이는 것을 확인할 수 있다. 또, 성능 면에서도 YAFFS나 FAT파일시스템에 비

표 3 마운트시간과 Heap 사용량 비교

	YAFFS	FAT	MNFS
마운트시간	6,441ms	220ms	185ms
Heap 메모리사용량	680Kbytes	N/A	34Kbytes

해 월등히 좋음을 알 수 있다.

MNFS의 마운트 시간과 Heap 메모리 사용량을 측정하는데, 시나리오 1을 수행한 상태, 즉 볼륨이 가득 찼을 때 볼륨을 마운트하는 시간과 Heap메모리 사용량은 표 3과 같다. FAT파일시스템의 경우 pSOS컴포넌트인 pHILE을 사용하기 때문에 Heap사용량을 측정할 수 없다. MNFS는 YAFFS에 비해 30배 이상 빠르게 마운트가 가능하고, Heap메모리도 1/20 밖에는 소모하지 않는다.

시나리오 5: 512바이트~16Kbytes까지 512바이트 단위로 크기를 늘려가며 쓰기요청에 대한 응답시간의 측정

마지막 실험으로 512바이트에서 16Kbytes까지 512바이트 단위로 증가시키며 쓰기요청에 대한 응답시간을 측정한다. 앞서 정의한 MNFS의 쓰기 시간에 대한 실험이다.

$$\left(\left(S_{page} \times t_{tr}\right) + t_{prog}\right) \times N$$

그림 12의 결과를 보면, 512바이트당 약 334us의 시간이 비례적으로 소모됨을 알 수 있다. 약간의 오차가 22회째와 29회째 쓰기 응답시간이 예측 값보다 적게 나왔으며, 오차는 6.8%와 4.1%로 매우 근소하다. 오차의 원인으로 수식에서 무시한 CPU연산 오버헤드와 NAND 페이지 기록시간 t_{prog}의 오차를 생각할 수 있다. 특히 t_{prog}는 데이타시트에 따르면 물리적인 NAND 플래시 메모리의 마모도와 상태에 따라 200us에서 최대 500us까지 걸림을 알 수 있다. 그림 12에서 세모표시 라인이 MNFS에 (512 * N) 바이트의 쓰기 요청에 대한 응답시간이다. 네모표시 라인은 (0.334 * N)의 값을 그래프로 표시한 것으로, 한 섹터에 대해 약 334us가 소모된다고 가정한 그래프다. 두 그래프가 거의 일치하는 것으로, MNFS가 연속된 쓰기 요청에 대해 예측 가능하게 동작함을 알 수 있다. 참고로, t_{prog}를 데이타시트상의 Typical 값인 200us로 가정할 경우, t_{tr}은 약 253ns이며, 이를 반영하면 512 * N 바이트의 쓰기 요청에 대해 MNFS의 쓰기 응답시간을 다음 식으로 예측할 수 있다.

$$\left((528 \times 0.253) + 200\right) \times N \quad (\text{us})$$

5. 결 론

MNFS는 모바일 멀티미디어 제품에 적합하도록 설계된 NAND플래시 메모리 기반의 파일시스템이다. 빠른 마운트 시간과 적은 자원의 소모, 그리고 순차적인 쓰기 요청에 대해 예측 가능한 쓰기 응답시간을 보장하는 것을 목표로 하였으며, 이를 통해 멀티미디어 스트리밍을 안정적으로 저장할 수 있도록 설계되었다. 특히 모바일 멀티미디어제품에서 요구되는 파일시스템 사용패턴을 분석하여, 적게 일어나는 갱신작업 대신 순차적인 읽기/쓰기 작업에 초점을 두어 설계하였으며, 이와 같은 접근은 매우 의미가 있다.

저장장치의 대부분을 차지하는 멀티미디어 파일의 경우 데이타 갱신이 필요한 경우가 적지만, DRM 정보나 재생목록과 같이 멀티미디어 파일을 다루기 위해 보조

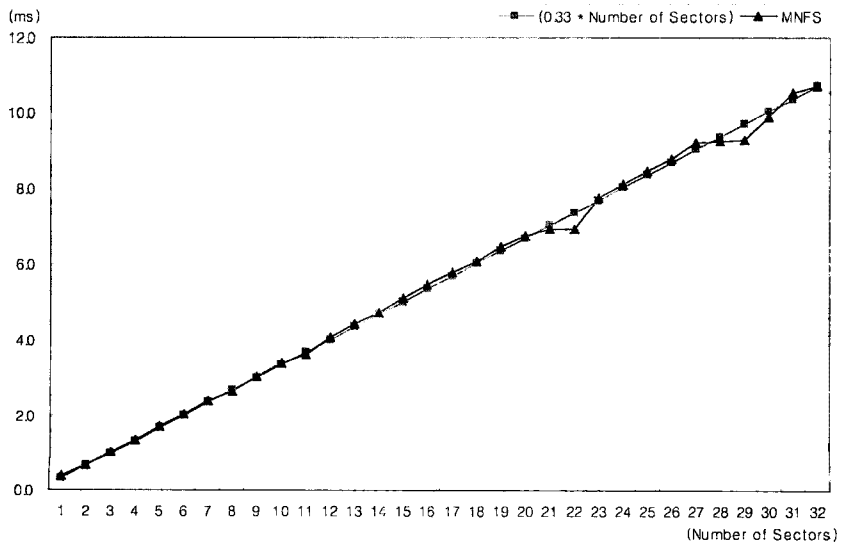

그림 12 쓰기 섹터개수에 따른 쓰기 응답시간

적으로 필요한 일부 정보의 경우 갱신작업이 부분적으로 필요하다. 이러한 파일을 MNFS에서 사용하면 성능 감소와 저장공간의 낭비를 초래할 우려가 있지만, 현재로서는 이러한 파일들을 위하여 별도의 파티션과 별도의 파일시스템을 추가로 사용하는 방법이 최선이다.

MNFS는 주기적으로 발생하는 스트리밍 데이타를 효과적으로 저장할 수 있는 모바일 멀티미디어 파일시스템으로서는 그 목표를 달성하지만, 그 과정에서 다양한 파일에 대한 배려를 하지 못하게 되었다는 문제점을 갖게 된다. 즉, 멀티미디어 파일이 아닌 파일을 저장하기에는 적합하지 못한 구조를 갖추었으며, 이러한 부분 때문에 실용성이 떨어지는 파일시스템이 되었다. MNFS가 연구목적의 파일시스템에서 벗어나 실제 제품에 사용될 수 있는 파일시스템이 되기 위해서는, 이러한 부분에 있어서의 보완이 필요하다.

참 고 문 헌

[1] W. Mangione-Smith, P. Ghang, S. Nazareth, P. Lettieri, W. Boring, R. Jain, "A low power architecture for wireless multimedia systems: lessons learned from building a power hog," Proceedings of the 1996 international symposium on Low power electronics and design, pp. 23-28.

[2] Samsung Electronics, "16M × 8 Bit NAND Flash Memory," http://www.samsung.com/.

[3] SSFDC Forum, "SmartMediaTM Specification," http://www.ssfdc.or.jp

[4] Samsung Electronics, "512 Mb/1Gb OneNANDTM Flash Memory," http://www.samsung.com/.

[5] Microsoft, "FAT: General Overview of On-Disk Format," Ver.1.03, 2000.

[6] Intel Corporation, "Understanding the flash translation layer (FTL) specification," http://developer.intel.com/.

[7] Li-Pin Chang, Tei-Wei Kuo, "An efficient management scheme for large-scale flash-memory storage systems," in Proceedings of the 2004 ACM symposium on Applied computing, 2004, pp. 862-868.

[8] D. Woodhouse, "JFFS: The Journaling Flash File System," in Ottawa Linux Symposium, 2001.

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

[10] M. Rosenblum, and J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System," ACM Transactions on Computer Systems, Vol. 10, No. 1, 1992, pp. 26-52.

[11] Li-Pin Chang, Tei-Wei Kuo, Shi-Wu Lo, "Real-time garbage collection for flash-memory storage systems of real-time embedded systems," in ACM Transactions on Embedded Computing Systems (TECS), Volume 3 Issue 4, 2004, pp. 837-863.

김 효 준

2006년 한양대학교 대학원 전자통신컴퓨터공학부 석사. 삼성전자 선임연구원 기술총괄 소프트웨어연구소. 관심분야는 파일 시스템

원 유 집

1990년 2월 서울대학교 계산통계학과 졸업. 1992년 2월 서울대학교 전산과학과 석사. 1997년 7월 Univ. of Minesota 졸업(박사). 1997년 9월~1999년 2월 Intel 연구원. 1999년 3월~현재 한양대학교 전자컴퓨터통신 공학부 부교수. 관심분야는 O/S and File system support for massive scale byte-addressable NVRAM, Large scale archiving and storage and file system support for multimedia application, Storage systems, Multimedia Networking, Internet Traffic Modeling and Analysis

김 요 환

2005년 한양대학교 전자컴퓨터공학부(학사). 2007년~현재 한양대학교 대학원 전자통신컴퓨터공학부 석사과정. 관심분야는 멀티미디 시스템, 파일 시스템