플래시 메모리를 위한 멀티미디어 파일 시스템의 구조 설계

양학모, 한용철, 류연승
명지대학교 컴퓨터소프트웨어학과
e-mail:ysryu@mju.ac.kr

A Study on Multimedia File System for Flash Memory

Hak-Mo Yang, Ryong-Cheol Han, Yeon-Seung Ryu
Dept. of Computer Software, Myongji University

요 약

플래시 메모리는 비휘발성 메모리로서 데이터 접근 속도가 빠르고 전력 소비가 적으며 가볍고 충격에 강한 특징을 가지고 있다. 최근 플래시 메모리의 가격이 저렴해지고 용량은 커져가고 있기 때문에 대용량의 멀티미디어 파일의 저장 장치로서 플래시 메모리의 사용이 증가할 것으로 보인다. 본 논문에서는 플래시 메모리를 위한 멀티미디어 파일 시스템의 구조 설계를 기술한다. 주요 특징으로는 i-node를 데이터 블록과 분리된 i-node 영역에 로그 방식으로 기록하고, 삭제 연산이 잦은 i-node 영역을 이동할 수 있게 하여 마모도 평준화를 고려하였다. 파일의 데이터 블록은 멀티미디어 응용 프로그램의 특징을 고려하여 인덱스화된 이중 연결 리스트 구조로 관리한다.

1. 서론

최근 디지털 카메라, PDA, 이동 전화기, USB 메모리 등의 데이터 저장 장치로서 플래시 메모리의 사용이 점차 증가하고 있다. 플래시 메모리는 EEPROM의 일종인 비휘발성 저장 매체로서 전원 공급이 단절되어도 데이터가 계속 남아있다. 또한, 하드 디스크와 비교하면 데이터 접근 속도가 빠르고, 전력 소비가 적으며, 가볍고 충격에 강한 장점을 가지고 있다. 표 1은 DRAM, 플래시 메모리, 하드 디스크의 데이터 접근 시간의 비교를 보여주고 있다.

플래시 메모리의 용량이 매우 빠르게 커져가고 가격은 저렴해지고 있기 때문에 대용량의 멀티미디어 파일의 저장 장치로서 플래시 메모리의 사용이 증가할 것으로 보인다. 그동안 FAT, JFFS2 등과 같은 파일 시스템이 쓰여 왔고 LFS(Log structured File System) 기반의 파일 시스템 등의 연구들이 있어 왔다. 그러나, 아직까지 멀티미디어 파일 시스템에 대한 연구 결과는 미미하다. 본 연구팀은 플래시 메모리의 특성을 고려하는 멀티미디어 파일 시스템을 연구하고 리눅스에서 개발하고 있다. 본 논문에서는 개발 중인 멀티미디어 파일 시스템의 구조 설계를 기술한다.

표 1 저장 매체의 데이터 접근 시간 비교

연산 / 매체	접근 시간		
	읽기	쓰기	삭제
DRAM	2.56 μs	2.56 μs	•
NOR Flash	14.4 μs	3.53 ms	1.2s (128KB)
NAND Flash	35.9 μs	226 μs	2ms (16KB)
Hard Disk	12.4 ms	12.4 ms	•

본 논문의 구성은 다음과 같다. 먼저, 2장에서는 플래시 메모리 및 멀티미디어 파일 시스템을 살펴보고, 3장에서는 멀티미디어 파일 시스템의 구조를 설

본 연구는 과기부 목적기초연구 (과제번호: R08-2004-000-10391-0) 지원으로 수행되었음

명한다. 마지막으로 4장에서 결론을 제시한다.

2. 멀티미디어 파일 시스템

2.1 플래시 메모리 특징

플래시 메모리는 블록들의 집합으로 구성되며, 각 블록은 고정된 개수의 페이지(page)들로 구성된다 (그림 1). 블록은 삭제 연산의 기본 단위이고 페이지는 읽기/쓰기의 기본 단위이다. 블록의 크기는 제품에 따라 4KB에서 128KB이며, 페이지의 크기는 1바이트에서 2KB이다. 각 페이지에는 여유 영역(spare area)이 있으며 시스템 데이터가 저장될 수 있다. 여유 영역은 보통 16바이트에서 64바이트의 크기를 가진다. 예를 들어, 삼성전자의 32MB NAND 플래시 메모리인 K9F5608X0B는 페이지 크기가 512바이트이며 여유 영역은 16바이트이다. 블록은 32개의 페이지로 구성되며 16KB 크기이다. 블록 내의 여유 영역은 512바이트가 된다. 이러한 블록이 2048개가 있어 32MB 용량이 된다.

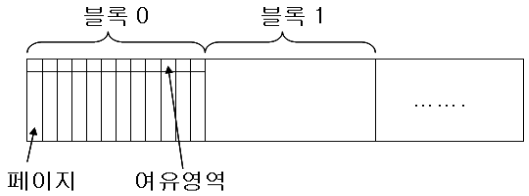

그림 1. 플래시 메모리의 구성

플래시 메모리는 하드 디스크와는 다른 특징을 가지고 있다. 첫째, 데이터 읽기/쓰기 단위와 삭제 단위가 다르다. 데이터 읽기/쓰기 단위는 페이지이고 삭제 단위는 블록이다. 둘째, 연산의 시간 차이가 크다. 표 1에서 NAND 플래시의 경우 쓰기가 읽기보다 7배 정도 느리다. 또한, 삭제 연산 시간은 읽기/쓰기보다 매우 느린 특징을 가지고 있다. 표 1을 보면 16KB(한 블록의 크기임)의 삭제 시간이 2ms이다. 세째, 데이터 쓰기 연산 시에 먼저 해당 페이지의 내용이 삭제(erase)되어야만 한다. 즉, 데이터가 쓰여진 위치에 데이터를 다시 기록하려면 그 영역을 삭제한 후에 기록해야 한다. 그런데, 삭제 단위는 블록이므로 페이지의 데이터를 변경하려면 페이지가 속해 있는 블록을 삭제한 후 변경해야 한다. 네째, 블록 삭제 연산의 횟수(erase cycle)에 제한이 있다. 일반적으로 십만 번에서 백만 번으로 제한되어 있으며 그 이상 삭제를 하게 되면 데이터의 무결성을 보장받지 못한다.

2.2 플래시 메모리의 소프트웨어 구조

플래시 메모리를 위한 시스템 소프트웨어는 운영체제의 파일 시스템과 FTL(Flash Translation Layer)이라 부르는 디바이스 드라이버이다. 90년대 중반 이후 플래시 파일 시스템에 대한 연구 결과가 발표되고 있다[1-5]. 리눅스의 JFFS, YAFFS를 비롯한 대부분의 플래시 파일 시스템들은 LFS (Log-structured File System)[7]의 개념을 사용하고 있다. LFS는 쓰기 연산 시에 메타 데이터와 실제 데이터를 병합하여 로그 영역의 블록들에 함께 기록한다. 하드 디스크의 경우 헤드의 탐색 시간을 감소시켜 쓰기 성능을 향상시킨다. 플래시 메모리에서는 데이터의 변경이 in-place-update 되지 않기 때문에 쓰기 성능을 향상시킬 수 있어 채택되고 있다. 그러나, 데이터가 변경되었을 때 무효화된 이전 데이터를 자유 공간으로 관리하기 위해 주기적으로 쓰레기 수집(garbage collection) 작업을 수행해야하는 문제점을 가진다.

FTL은 파일 시스템에게 블록 디바이스 드라이버의 인터페이스를 제공하는 소프트웨어로서 운영체제 내의 디바이스 드라이버 또는 플래시 디스크의 컨트롤러 내의 펌웨어로 구현된다[6]. 리눅스에서는 MTD 수준에서 FTL 드라이버가 제공된다. FTL은 파일 시스템의 논리 블록 주소와 연산(읽기/쓰기) 요청을 받아 실제 물리 블록 주소에 대한 연산을 수행해준다.

2.3 멀티미디어 파일 시스템

멀티미디어 파일은 용량이 크고 주로 읽기 전용이며 실시간 입출력을 요구하는 특징을 가진다. 멀티미디어 파일 시스템은 주로 하드 디스크 기반의 저장장치에서 연구되어왔다[9,10]. 이러한 연구들은 멀티미디어 데이터의 실시간 처리를 위한 디스크 헤드 스케줄링 알고리즘 및 데이터 블록의 할당 정책을 연구하였다. 또한, 디스크 입출력 대역폭, 메모리 버퍼의 자원 제약 조건을 고려하는 수용 제어 알고리즘들이 연구되었다.

플래시 메모리 저장 장치에서 멀티미디어 파일 시스템에 대한 연구 결과는 아직까지 미미하다. 본 연구팀은 플래시 메모리의 특성을 고려하는 멀티미디어 파일 시스템을 연구하여 리눅스에서 설계 개발하고 있다.

(1) 파일 시스템의 구조 설계 : 수퍼 블록, 가용 블록의 관리, 파일의 메타 데이터, 파일 및 디렉토리

의 블록 할당 방법 등을 연구한다.

(2) 파일 연산 : 마운트, 파일 및 디렉토리의 생성, 삭제, 읽기, 쓰기 등의 연산을 연구한다.

(3) 실시간 입출력 : 파일의 실시간 스트리밍을 보장하기 위해 수용 제어 기법을 연구한다.

본 논문에서는 파일 시스템의 구조 설계 중 메타 데이터의 구조 및 블록 할당 방법을 다룬다.

3. 파일 시스템 구조 설계

연구된 멀티미디어 파일 시스템은 리눅스의 파일 시스템에서 사용하는 데이터 구조인 i-node, 디렉토리 등을 도입하여 리눅스 파일 시스템의 철학을 유지한다. 그러나, i-node 영역과 데이터 블록 영역을 분리하고 i-node 영역의 위치 정보를 가지는 i-node 맵을 두었다. 또한, i-node의 변경은 저널 파일 시스템과 비슷한 로그 방식으로 수행된다.

파일 시스템은 파일이 접근될 때마다 접근 시간을 기록해야 하고 파일에 데이터가 기록될 때 데이터 블록의 주소를 기록해야 한다. 이때, i-node에 그 정보가 기록 또는 변경된다. 일반적인 유닉스 파일 시스템에 대한 워크로드를 분석해보면 쓰기 접근의 60-70%가 i-node와 같은 메타 데이터로 집중되고 있음이 알려져 있다[8].

멀티미디어 파일은 거의 변경이 없는 읽기 전용 데이터인데 쓰기 및 변경이 빈번한 i-node를 데이터와 같은 블록에 저장하는 경우 변경으로 인해 무효화된 i-node들을 관리하여 쓰레기 수집을 해야 하는 부담이 커지게 된다. 따라서, 본 연구에서는 i-node 영역을 파일 데이터 영역과 분리하였다.

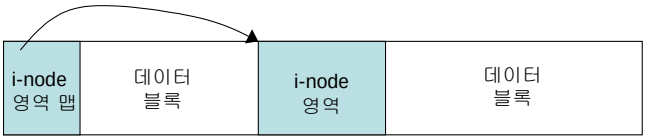

그림 2. i-node 영역과 i-node 영역 맵

(1) i-node

i-node는 ext2 파일 시스템의 i-node처럼 파일의 정보를 가지며 128 바이트의 크기라는 점에서 거의 비슷하지만 파일의 데이터 블록의 관리가 다르다. 각 i-node는 고유한 i-node 번호를 갖는다.

(2) i-node 영역

i-node 영역은 i-node를 저장하기 위해 예약된 연속된 블록들의 집합이다. i-node 영역의 블록 수는 고정 값으로서 파일 시스템을 초기화할 때 결정된다. i-node 영역의 위치는 필요에 따라 변경될 수 있으며 그 시작 위치는 i-node 영역 맵에 기록된다.

플래시 메모리 블록의 페이지가 512 바이트인 경우 한 페이지에 i-node 네 개가 저장된다. 그림 3은 i-node 영역의 한 블록을 보여주고 있다. 블록 내 각 페이지에는 i-node의 번호가 연속되고 4로 나눈 몫이 같은 i-node들이 함께 저장된다. 만약 페이지 크기가 2KB인 경우에는 각 페이지에 i-node 번호가 연속되고 8로 나눈 몫이 같은 i-node들이 저장된다. 페이지의 몫 값은 여유 영역에 기록되어 i-node 영역에서 i-node를 찾을 때 사용된다. 예를 들어, 번호가 103인 i-node를 찾는다면 여유 영역에서 몫 값이 25인 페이지를 찾고 그 페이지에서 네 번째 i-node를 찾으면 된다.

	데이터 영역				여유 영역
페이지 0	i-node 0	i-node 1	i-node 2	i-node 3	0 -
페이지 1	i-node 4	i-node 5	i-node 6	i-node 7	1 -
페이지 2	i-node 20			i-node 23	5 -
페이지 3		i-node 1			1 0
페이지 4		i-node 1			1 0

그림 3. i-node 영역의 예

각 페이지의 여유 영역에는 i-node의 몫 값, 페이지 내의 각 i-node 엔트리들의 상태 (used/free/invalid), i-node의 복사본이 있는 경우에 이전 복사본이 있는 페이지의 번호 등이 기록된다.

파일이 새로 생성되어 i-node가 새로 만들어지는 경우에는 같은 몫 값을 가지는 페이지가 이미 존재하는 지를 검사한다. 만약 그런 페이지가 없다면 가용 페이지를 찾아 i-node를 기록한다. 그림 3에서 i-node 20이 생성될 때 가용 페이지가 2번이었다면 그 페이지에 첫 번째 엔트리에 기록된다. 또, i-node 23이 생성되는 경우에는 몫 값이 5인 페이지인 2번 페이지를 찾아 네 번째 엔트리에 기록된다.

어떤 i-node의 내용이 변경되면 원래 위치에 변경되지 않고 새로운 페이지에 기록된다. 그림 3에서 1번 i-node가 변경되는 예를 보이고 있다. 처음 변경되면 가용 페이지인 3번 페이지의 두 번째 엔트리에 기록된다. 이때, 여유 영역에는 i-node 번호의 몫 값 1과 원래 i-node가 저장되어 있던 페이지의 번호인 0을 기록한다. 1번 i-node가 또 변경되어 가용 페이

지인 4번 페이지에 기록되었다.

i-node 영역의 모든 페이지가 사용되어 더 이상 가용 블록이 없다면 i-node 영역의 쓰레기 수집 작업을 수행한다. 쓰레기 수집 횟수가 특정 한도를 넘게 되면 마모도 평준화를 고려하여 i-node 영역을 새로운 위치로 변경한다. 새 영역의 시작 블록 번호는 i-node 영역 맵에 기록한다.

(3) i-node 영역 맵

i-node 영역 맵은 i-node 영역의 시작 블록 번호 값을 저장하고 있는 블록이다. i-node 영역 맵의 위치는 고정되어 변하지 않는다. i-node 영역의 시작 블록 번호 값은 로깅 방식으로 기록된다. 즉, 시작 블록 번호 값이 변경되면 원래 위치에 변경하지 않고 다음 엔트리 위치에 기록한다. i-node 영역의 위치는 자주 바뀌지 않으므로 i-node 영역 맵은 빨리 차지 않는다. 만약 다 차게 되면 블록을 삭제하고 처음 위치부터 값을 저장해나간다.

(4) 디렉토리

디렉토리는 디렉토리 엔트리들로 구성된다. 디렉토리 엔트리는 252 바이트 크기의 파일 이름과 4 바이트 크기의 i-node 번호로 구성된다.

(5) 데이터 블록

다음 사항을 고려하였다. 첫째, 동영상 파일은 수 MB 이상으로 용량이 크기 때문에 데이터 블록의 할당 단위를 크게 한다. 둘째, 동영상 플레이어 프로그램들은 상영(playback)을 할 때 주로 순차적인 블록 읽기를 하지만, forward, backward, jump 등의 VCR 기능을 제공한다.

데이터 블록의 할당 단위는 플래시 메모리에서 삭제 단위인 블록으로 한다. 새로운 데이터 블록은 임의의 블록을 할당하며 VCR 연산을 고려하여 한 파일에 속한 데이터 블록들을 순차적인 이중 연결 리스트(doubly linked list)로 관리한다(그림 4). 또한, 파일의 임의 접근을 위해 i-node에서 파일 데이터 블록에 대한 sparse 인덱스 테이블을 관리한다.

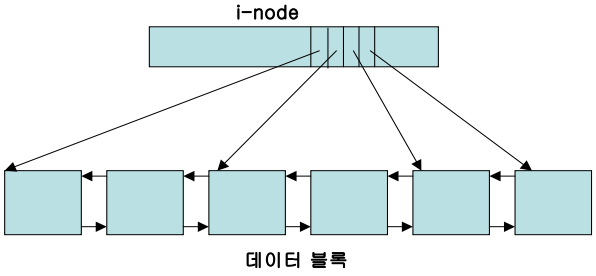

그림 4. i-node와 파일 데이터 블록

데이터 블록은 마모도 평준화를 고려하여 i-node 영역과 교환되어 다른 위치로 이동될 수 있다.

4. 결론

본 연구에서는 플래시 메모리 저장 장치를 위한 멀티미디어 파일 시스템의 구조를 설계하였다. 주요 특징으로는 i-node를 데이터 블록과 분리된 i-node 영역에 로그 방식으로 기록하고, 삭제 연산이 잦은 i-node 영역을 이동할 수 있게 하여 마모도 평준화를 고려하였다. 파일의 데이터 블록은 멀티미디어 응용 프로그램의 특징을 고려하여 인덱스화된 이중 연결 리스트 구조로 관리된다. 본 연구에서 설계 중인 멀티미디어 파일 시스템은 리눅스에서 구현 중이며 플래시 파일 시스템인 JFFS2, YAFFS와 성능 비교 연구를 수행할 계획이다.

참고문헌

[1] A. Kawaguchi, S. Nishioka, and H. Motoda, "A Flash-Memory Based File System," *Proceedings of the 1995 USENIX Technical Conference*, Jan. 1995.

[2] P. Torelli, "The Microsoft Flash File System," *Dr. Dobb's Journal*, Feb. 1995.

[3] M. Wu and W. Zwaenepoel, "eNVy: A Non-Volatile, Main Memory Storage System," *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1994.

[4] F. Douglis, R. Caceres, F. Kaashoek, K. Li, B.Marsh, and J. A. Tauber, "Storage Alternatives for Mobile Computers," *Proceedings of the 1st Symposium on OSDI*, 1994.

[5] M. L. Chiang, Paul C. H. Lee, and R. C. Chang, "Using Data Clustering to Improve Cleaning Performance for Flash Memory," *Software Practice & Experience*, Vol. 29, No.3, pp. 267-290, Mar. 1999.

[6] Intel Corporation, "Understanding the Flash Translation Layer Specification," http://developer.intel.com

[7] M. Resenblum and J. Ousterhout, "The Design and Implementation of a Log-structured File System," *ACM Trans. on Computer Systems*, Vol. 10, No. 1, Feb. 1992.

[8] C. Ruemmler and J. Wilke, "UNIX Disk Access Patterns," *Proceedings of 1993 Winter USENIX Conference*, Jan. 1993.

[9] B. Ozden, R. Rastogi, and A. Silberschatz, A Framework for the Storage and Retrieval of Continuous Media Data, *Proc. of the IEEE International Conference on Multimedia Computing and Systems*, May. 1995

[10] D. J. Gemmel, H. M. Vin, D. D. Kandler, P. V. Rangan, and L. A. Rowe, "Multimedia Storage Servers : A Tutorial," *IEEE Computer*, pp. 40-49, May. 1995.