플래시메모리 기반 저장장치의 설계기법

임근수 고 건
서울대학교 컴퓨터공학부
{ksyim, kernkoh}@oslab.snu.ac.kr

A Study on Flash Memory Based Storage Systems Depending on Design Techniques

Keun Soo Yim Kern Koh
School of Computer Science and Engineering, Seoul National University

요 약

최근 모바일 컴퓨팅환경으로 컴퓨팅 패러다임이 전이함에 따라 휴대가 용이하고 저전력으로 동작하는 플래시메모리가 차세대 저장매체로 부상하고 있다. 하지만 플래시메모리를 저장장치로 활용하기 위해서는 몇 가지 설계상의 제약이 있다. 대표적으로 데이터를 쓰기 위해서는 비교적 수행시간이 길고 큰 단위로 이뤄지는 삭제연산을 선행해야 한다는 것이다. 이런 제약을 극복하고 저장장치를 구성하는 방법에는 크게 미들웨어를 활용하는 방법과 전용 파일시스템을 사용하는 방법이 있다. 본 논문에서는 이 두 가지 설계기법의 구조와 세부특성에 대해 살펴본다. 이를 통해 플래시메모리 기반 저장장치를 구성하는데 있어서 고려해야 할 성능상의 요소에 대해 고찰하고 향후 과제를 제시한다.

1. 서론

최근 컴퓨팅 환경에 대한 패러다임 자체가 기존의 데스크 탑 중심에서 언제 어디서나 원하는 정보와 컴퓨팅 파워를 사용할 수 있게 하는 유비쿼터스 컴퓨팅으로 전환되고 있으며, 이로 인해 휴대폰, PDA, 디지털카메라 등과 같은 휴대용 정보기기의 사용이 급증하고 있다. 이와 같은 휴대용 기기에서는 일반적으로 하드디스크 대신 플래시메모리를 보조기억장치로 사용한다. 왜냐하면 플래시메모리는 가볍고 물리적인 충격에 강해 휴대가 용이하고 전력을 적게 소모해 동일전력으로 장시간 사용할 수 있기 때문이다[1].

플래시메모리는 일종의 *EEPROM(Electrically Erasable and Programmable ROM)*으로 크게 바이트 I/O를 지원하는 NOR형과 페이지 I/O만을 지원하는 NAND형이 있다. NOR형 플래시메모리는 읽기 속도가 빠른데 반하여 쓰기 속도가 느려 주로 코드용 메모리로 사용하며[2], NAND형 플래시메모리는 쓰기 속도가 빠르고 단위 공간당 단가가 낮아 주로 대용량 데이터 저장장치로 사용한다[3]. 표 1은 플래시메모리와 다른 메모리 소자와의 성능상의 특성을 비교한 것으로 비교적 낮은 단가로 빠른 읽기속도를 제공하는 영속성 있는 저장장치임을 보인다.

하지만 EEPROM인 플래시메모리에서 데이터를 쓰기 위해서는 삭제연산을 선행해야 하며 삭제 단위는 쓰기 단위보다 크다. 이러한 특성은 플래시메모리를 주 메모리로 사용하는 것을 어렵게 하고, 보조기억장치로 사용하는 경우에도 일반 하드디스크용 파일시스템을 그대로 활용하는 것을 저해한다. 그래서 삭제연산을 감추기 위해서 파일시스템과 플래시메모리 사이의 미들웨어인 플래시 변환계층(Flash Translation Layer, FTL)이 제안되어 사용하고 있다[4, 5, 6, 7]. FTL은 쓰기연산 시에 파일시스템이 생성한 논리주소를 플래시메모리상의 이미 삭제연산을 수행한 영역에 대한 물리주소로 변환하는 역할을 수행한다. 빠른 주소변환을 위해 주소변환 테이블은 비교적 단가가 높은 SRAM을 사용해 구성한다. FTL을 사용함으로써 호스트에서는 FAT와 같은 일반적인 자기디스크용 파일시스템으로도 플래시메모리를 효율적으로 제어할 수 있다.

표 1. 다양한 메모리 소자의 특성비교.

	Read	Write	Erase	Cost/MB
DRAM	60ns (2B) 2.6μs (512B)	60ns (2B) 2.6μs (512B)	-	30~40
NOR-type Flash	150ns (1B) 15μs (512B)	211μs (1B) 3.5ms (512B)	1.2s (128KB)	20~30
NAND-type Flash	10μs (1B) 36μs (512B)	226μs (1B) 266μs (512B)	2ms (16KB)	10~20
Disk	12.4ms (512B)	12.4ms (512B)	-	1

하지만 플래시메모리는 실시간 데이터베이스와 같은 특수한 응용을 제외하곤 주 메모리가 아닌 보조기억장치로 사용한다. 이 경우 FTL을 사용하지 않고도 Log-Structured File System (LFS)와 같은 구조의 파일시스템을 활용해 플래시메모리의 성능을 개선할 수 있다[8, 9, 11, 12]. 전용 파일시스템을 사용하는 경우에는 FTL과 관련된 부과 하드웨어를 줄일 수 있다는 장점이 있다.

본 논문의 2장에서 플래시메모리의 종류와 세부구조에 대해 살펴본 후, 3장과 4장에서 각각 앞서 언급한 두 가지 플래시메모리 기반 저장장치의 구성기법에 대해 기술한다. 3장에서는 FTL에 기반한 방식의 구조와 다양한 설계기법들에 대해 살펴보고, 4장에서는 플래시메모리 전용 파일시스템을 사용하는 방식의 구조와 세부작동원리에 대해 알아본다. 그리고 5장에서는 이 두 가지 기법에 대해 고찰한 후 결론을 맺고 향후 연구과제를 제시한다.

2. 플래시메모리의 종류 및 세부구조

일종의 EEPROM인 플래시메모리의 셀 구조는 그림 1과 같다. 소스(Source)와 드레인(Drain)의 연결상태를 중앙 상단에 위치한 제어 게이트(Control Gate)와 하단에 위치한 부동 게이트(Floating Gate)를 사용해 제어한다. 추가적으로 제어 게이트가 있는 것을 제외하고는 부동 게이트에 일정 값 이상의 전압을 인가함으로써 소스와 드레인 사이에 전기적 채널을 형성해 소스의 값을 드레인으

로 전달하는 일반적인 MOS-FET 트렌지스터와 유사한 구조이다[20]. 그림상의 흰색과 회색 부분은 각각 p형과 n형 반도체로 도핑된 부분을 의미한다. 즉, NAND형은 pMOS와 NOR형은 nMOS와 유사한 형태로 도핑된다.

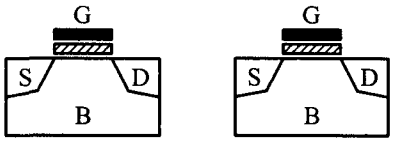

그림 1. 플래시메모리의 셀 구조. (좌: NAND형, 우: NOR형)

NAND형 플래시메모리는 초기에 부동 게이트에 양이온을 충전하고 있어 논리적 '1'값을 나타낸다. 셀 값은 소스에 전압을 인가한 후 드레인의 값을 확인함으로써 읽을 수 있다. 만약 부동 게이트의 값이 논리적 '1'이면 소스와 드레인 사이의 베이스(Base)영역에 음이온 채널이 형성되어 소스의 값이 드레인에 전달되고, 그렇지 않으면 소스와 드레인은 전기적으로 연결되지 않아 개방(open)상태가 된다. 프로그램 시에는 제어 게이트에 높은 전압을 인가함으로써 베이스의 음전하를 부동 게이트에 축적해 부동 게이트의 값을 논리적 '0'으로 설정한다. 그리고 삭제연산은 이와는 반대로 게이트를 접지시키고 베이스에 고압을 인가해 부동 게이트에 축적된 음전하를 방출시킨다. 이때 삭제 단위(블록 또는 세그먼트)는 인접한 페이지(읽기 및 쓰기 단위)의 집합으로 플래시메모리의 종류와 내부구조에 따라 수행시간이 길게는 1~2초에 달하며, 삭제연산을 수행할 수 있는 총 횟수에 제한이 있다.

NOR형 플래시메모리는 이와는 반대로 게이트에 전압이 걸리게 되면 베이스의 p형 반도체와 서로 척력이 작용해 소스와 드레인 사이가 개방되며, 0V가 걸리는 경우에 채널이 형성돼 전류가 흐르게 된다. 따라서 NOR형 셀은 초기에는 부동 게이트에 논리적 '0' 값을 유지하고 프로그램을 통해 논리적 '1'값을 저장하고 삭제연산을 통해 다시 값을 '0'으로 복원한다.

NAND형과 NOR형 플래시메모리는 위의 셀을 사용해 전체 메모리시스템을 구성하는 단계에서 그 차이가 확연하게 드러난다. NAND형의 경우 그림 2(좌)와 같이 셀을 배치하고, 진하게 표시된 셀들의 값은 W/L(word line) i를 접지시키고 이를 제외한 B/L(bit line), SSL, GSL, W/L 노드들에 특정전압을 인가한 후 B/L에 걸리는 전압을 확인함으로써 읽을 수 있다. 만약 W/L i에 해당하는 셀의 값이 '1'이라면 셀 내부의 소스와 드레인이 연결되어 B/L은 접지된 것이 돼 전압이 떨어진다.

NOR형은 그림 2(우)와 같은 구조를 가지며 진하게 표시된 셀의 값은 B/L과 W/L 3에 전압을 인가하고 이외의 W/L과 C 노드를 접지시켜 B/L의 값을 확인함으로써 읽을 수 있다. 만약 선택된 W/L 3의 부동 게이트에 축적된 값이 '0'이라면 해당 셀의 소스와 드레인이 연결되어 B/L은 접지 노드 C와 연결돼 전압이 떨어지며, 축적된 값이 '1'이라면 B/L은 인가된 전압을 유지한다.

위와 같은 구조적 차이로 인해 NOR형은 바이트 I/O를 지원하며 NAND형은 페이지(512 또는 2048바이트) I/O만은 지원한다. 상대적으로 NOR형 플래시메모리는 읽기 속도가 빠른데 반하여 쓰기 속도가 느려 주로 코드용 메모리로 사용하며[2], NAND형 플래시메모리는 쓰기 속도가 빠르고 단위 공간당 단가가 낮아 주로 대용량 데이터 저장장치로 사용한다[3].

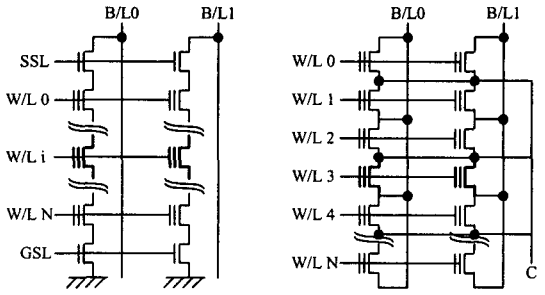

그림 2. NAND형(좌)과 NOR형(우) 플래시메모리의 구조.

3. 플래시 변환계층 (Flash Translation Layer, FTL)

서론에서 소개한 것과 같이 FTL은 플래시메모리의 삭제연산을 감추기 위한 미들웨어로 파일시스템과 플래시메모리 사이에 위치한다. 삭제연산은 쓰기연산 시에 파일시스템이 생성한 논리주소를 플래시메모리상의 이미 삭제연산을 수행한 영역에 대한 물리주소로 변환함으로써 감춰진다. 삭제연산을 감추고 I/O를 하나의 단위(atomic operation)로 처리해 하드디스크와 같은 단일 저장장치(storage system)를 구성함으로써, 상단에서 일반 파일시스템을 사용해 플래시메모리를 제어할 수 있다. 이는 현대의 운영체제의 기본 철학인 정책(policy)과 기능(mechanism)을 각각 파일시스템과 저장장치로 구분한 것으로 볼 수 있다. 기능에 해당하는 FTL은 크게 호스트시스템에서 독립된 하드웨어 형태[4, 5, 6]와 호스트시스템 내부의 디바이스드라이버 형태[7]로 구현할 수 있다.

FTL은 주소변환 단위에 따라 크게 페이지(쓰기)단위 주소변환(page-level address mapping)과 블록(삭제)단위 주소변환(block-level address mapping)으로 나뉜다. 페이지단위 주소변환 – 그림 3(a) – 은 정교하게 주소를 변환하기 때문에 성능은 좋은데 반하여 주소변환 테이블의 크기가 커 제작비용이 높다. 반대로 블록단위 주소변환 – 그림 3(b) – 은 비정교하게 주소를 변환하기 때문에 주소변환 테이블의 크기는 작지만, 내부의 하나의 페이지에 대한 수정연산이 발생해도 전체 블록을 삭제하고 갱신해야 하는 추가비용이 있다. 뿐만 아니라 이 과정을 수행하는 도중에 폴트가 발생하면 데이터의 일관성을 깨뜨릴 수 있다.

이러한 블록단위 주소변환 방식의 단점을 개선한 기법은 교체블록 기법(replacement block scheme) – 그림 3(c) – 으로 블록 내부의 페이지에 대한 갱신요청이 발생하면 교체블록을 할당해 쓰기를 수행하고 이를 연결리스트로 구성해 향후 읽기연산 시 이 리스트를 역순으로 검색해 최종적으로 수정된 데이터를 제공하는 방법이다.

마지막으로 최근에 제안된 로그블록 기법(log-block scheme) – 그림 3(d) – 은 페이지단위 주소변환과 블록단위 주소변환 기법을 병합한 형태로 비교적 큰 단위로 요청되는 순차 입출력은 블록단위로 처리하고, 작은 단위로 요청되는 임의 입출력은 페이지 단위로 로그구조 파일시스템(Log-Structured File System, LFS)[11, 12]과 유사하게 로그형태로 저장하는 방식이다[6]. 이를 통해 주소변환 테이블의 크기를 줄이고도 높은 성능을 발휘할 수 있다.

하지만 FTL을 사용하여도 호스트시스템에서는 FAT와 같은 일반적인 자기디스크용 파일시스템을 사용해야 하며, 하나의 작업을 두 개의 층을 두어 처리하는 구조가 된다. 초기에 FTL이 제안된

배경은 플래시메모리를 실시간 데이터베이스와 같은 특수한 응용에서 주 메모리로 활용하기 위한 것이었다[4]. 따라서 NAND형 플래시메모리와 같이 보조기억장치로 사용하는 경우에는 FTL을 활용해 저장장치를 설계하는 것이 플래시메모리 전용 파일시스템만을 사용해 설계하는 것에 비해 비효율적일 수 있다. 뿐만 아니라 FTL과 관련된 기술은 이미 전세계적으로 특허등록이 되어있어 PCMCIA 카드에 기반하지 않은 형태의 사용에는 법률상의 제약이 따른다.

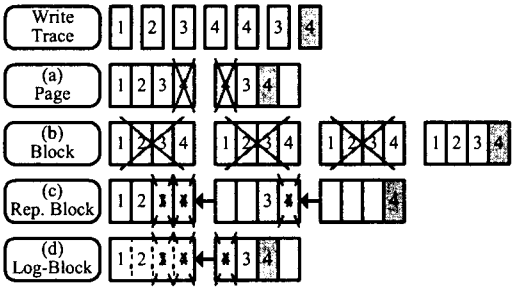

그림 3. FTL의 설계기법에 따른 작동예제.

4. 플래시메모리 전용 파일시스템

이러한 문제점을 개선해 추가비용 없이 파일시스템에서 효율적으로 플래시메모리를 제어하기 위해서 JFFS(Journaling Flash File System)과 FFS(Flash File System)과 같은 플래시메모리 전용 파일시스템이 개발되었다[8, 9, 13, 14]. JFFS는 GNU GPL[15] 아래 공포되어 주로 리눅스 기반 기기에서 사용하며, 비교적 간단한 구조의 FFS는 MS-DOS에서 사용한다.

JFFS[8, 9, 13]는 그 이름에서 알 수 있듯이 LFS과 유사하게 쓰기의 경우에는 데이터를 로그형태로 순차적으로 기록하고 읽기의 경우에는 로그를 역순으로 검색해 가장 최신의 데이터를 읽는다. 쓰기연산 이후에 임계용량 이상을 사용하게 되면 앞부분 블록부터 차례로 삭제연산(garbage collection)을 수행해 빈 공간을 확보한다. 이때 삭제 블록내부에 존재하는 유효한 페이지는 다시 로그형태로 기록해 보존한다.

하지만 이와 같이 삭제연산을 순차적으로 수행하는 것은 유효한 데이터의 비율 등을 고려해 최적의 블록을 선택적으로 삭제하는 방법에 비하여 비효율적이다. 이를 개선하기 위해 JFFS2[14]가 개발되었다. 삭제연산을 효율적으로 수행하는 것 외에도 JFFS2에서는 단위공간당 단가가 높은 플래시메모리의 공간을 효율적으로 활용하기 위해 압축기능을 사용하고[10], 보다 다양한 종류의 inode 구조를 지원해 몇 가지 장점을 취한다.

플래시메모리 전용 파일시스템과 FTL을 설계하는데 있어서 성능에 큰 영향을 끼치는 설계 요소는 '언제 어떤 블록을 삭제할 것인가'라는 정책결정 과정이다. 너무 일찍 삭제연산을 수행하면 삭제 블록내부에 유효한 페이지의 수가 적어 총 수행하는 삭제연산의 수가 늘어나게 된다. 반대로 삭제연산 수행시점을 최대한 지연시키면 공간 효율성 측면에서는 좋지만 이후에 쓰기연산이 요청되고 기 삭제된 블록이 없는 경우에 그 동안 지연시킨 삭제연산을 한번에 수행해야 하기 때문에 입출력 성능을 크게 저하시킬 수 있다[4]. 왜냐하면 최근에 플래시메모리 중에는 내부를 뱅크(bank)형태로 분할하고 서로 다른 뱅크에 대해 삭제와 쓰기 또는 쓰기와 읽기연산을 동시에 수행할 수 있는 기능을 지원하는데, 삭제연산을 지연해 특정시점에 동시에 수행하게 되면 이러한 구조적 병렬성을 활용하기 어렵기 때문이다. 그리고 삭제할 블록을 선택하는 문제는 LFS의 기법을 응용할 수 있다[11, 12].

5. 결론

본 논문에서는 유비쿼터스 및 모바일 컴퓨팅 환경에서 차세대 저장매체로 떠오르고 있는 플래시메모리에 기반한 저장장치를 구성하는 두 가지 기법에 대해 살펴보았다. 두 가지 방식이 각각 상대적인 장단점을 갖기 때문에, 앞으로 연구에서는 응용에 적절한 기법을 선택해 삭제연산과 같은 성능을 개선해 나가는 일이 요구된다. 향후 연구에서는 기 제작된 시뮬레이터[16, 17]와 벤치마크들[18, 19]을 성능평가 단계에서 활용할 수 있다.

참고 문헌

[1] 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 System Design and Implementation*, pp. 25-37, 1994.

[2] Intel Corporation, "3 Volt Synchronous Intel StrataFlash Memory," http://www.intel.com/.

[3] Samsung Electronics, "128M x 8 Bit / 64M x 16 Bit NAND Flash Memory," http://www.samsungelectronics.com/.

[4] M. Wu and W. Zwaenepoel, "eNVy: A Non-Volatile, Main Memory Stroage System," *In Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 86-97, 1994.

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

[6] J. Kim, J.M. Kim, S.H. Noh, S.L. Min, and Y. Cho, "A Space-Efficient Flash Translation Layer for CompactFlash Systems," *IEEE Transactions on Consumer Electronics*, Vol. 48, No. 2, pp.366-375, 2002.

[7] A. Kawaguchi, S. Nishioka, and H. Motoda, "A Flash Memory Based File System," *In Proceedings of the USENIX 1995 Winter Technical Conference*, pp. 155-164, 1995.

[8] D. Woodhouse, "JFFS: The Journaling Flash File System," Ottawa Linux Symposium (http://sources.redhat.com/jffs2/), 2001.

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

[10] K.S. Yim, H. Bahn, and K. Koh, "A Compressed Page Management Scheme for NAND-type Flash Memory," *In Proceedings of the International Conference on VLSI*, pp. 266-271, 2003.

[11] 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, pp. 26-52, 1992.

[12] M. Seltzer, K. Bostic, M.K. McKusick, and C. Staelin, "An Implementation of a Log-Structured File System for UNIX," *In Proceedings of the 1993 Winter USENIX Technical Conference*, pp. 307-326, 1993.

[13] JFFS, http://developer.axis.com/software/jffs/.

[14] JFFS2, http://sources.redhat.com/jffs2/.

[15] GNU GPL, http://www.gnu.org/licenses/gpl.html/.

[16] A. Rubini and J. Corbet, *Linux Device Driver*, 2nd Ed., O'Reilly, 2001.

[17] 정재용, 노삼혁, 민상렬, 조유근, "플래시메모리 시뮬레이터의 설계 및 구현," 한국정보과학회 논문지, 컴퓨팅의 실제, Vol. 8, No. 1, pp. 36-45, 2002.

[18] R. Arnold and T. Bell, "A Corpus for the Evaluation of Lossless Compression Algorithms," *In Proceedings of the IEEE Data Compression Conference*, pp. 201-210, 1997.

[19] Andrew Benchmark, ftp://ftp.cs.cmu.edu/user/satya/ab/.

[20] N.H.E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, 2nd Ed., Addison-Wesley, 1994.