FAT 호환 플래시 메모리 파일 시스템을 위한 성능 최적화 기법

김성관†○ 이동희‡ 민상렬†
†서울대학교 전기컴퓨터공학부, {skkim, symin}@archi.snu.ac.kr
‡서울시립대학교 컴퓨터과학부, dhlee@venus.uos.ac.kr

KISS Korea Computer Congress 2005

Sung-Kwan Kim†○ Dong-Hee Lee‡ Sang Lyul Min†
†School of Computer Science and Engineering, Seoul National University
‡Department of Computer Science, University of Seoul

요 약

최근 휴대용 단말기 및 임베디드 시스템 등을 위한 저장 장치로서 플래시 메모리가 각광받고 있다. 이 때 저장 장치 관리를 위한 플래시 메모리 파일 시스템의 역할이 중요한데, 이를 위해 전용 플래시 메모리 파일 시스템이 제안되기도 했지만 산업계에서는 사실상의 표준으로 FAT 파일 시스템이 광범위하게 사용되고 있다. 그러나 FAT 파일 시스템은 플래시 메모리의 물리적 특성을 고려하지 않고 설계되었기에 성능상 개선의 여지가 있다.

본 논문에서는 FAT 파일 시스템을 대상으로 플래시 메모리의 동작 특성을 고려한 성능 최적화 기법을 제안한다. 구체적으로 본 논문에서 제안되는 기법은 파일 삭제 시 FAT 파일 시스템의 기본 동작을 확장한 것으로 플래시 메모리 위에서 동작하는 FAT 파일 시스템의 쓰기 성능을 개선하는 효과를 보여 준다. 실험 결과 약 29%의 쓰기 성능 개선 효과가 있음을 확인할 수 있었다.

1. 서 론

플래시 메모리는 비휘발성의 메모리 반도체로 집적도가 높고 충격에 강하며 저전력으로도 동작 가능하기 때문에 최근 휴대용 단말기 및 임베디드 시스템 등을 위한 저장 매체로서 각광받고 있다. 주로 사용되는 플래시 메모리에는 NOR형과 NAND형의 두 가지 종류가 있는데, 바이트 단위로 주소 지정이 가능한 NOR형은 실행 코드 저장용으로, 페이지 단위 또는 블록 단위로만 주소 지정이 가능한 NAND형은 데이터 저장용으로 주로 사용되고 있다.

플래시 메모리는 하드디스크와 같은 일반적인 블록 장치와는 달리 읽기(read), 쓰기(write) 동작 이외에 소거(erase) 동작이 필요하다는 큰 차이점이 있다. 즉, 저장되어 있는 기존의 데이터를 새로운 값으로 갱신하기 위해서는 그 위치에 해당하는 메모리 셀(cell)을 먼저 소거한 다음, 새 데이터 값을 기록해야만 한다. 따라서, 이러한 동작 특성을 숨기고 일반적인 파일 시스템이 필요로 하는 섹터 읽기 및 쓰기 인터페이스를 제공해 주기 위하여 특별한 소프트웨어 모듈을 사용하게 되는데 이를 보통 FTL(Flash Translation Layer)이라 부른다[1].

그림 1-(A)는 이렇게 FTL 모듈이 사용되는 경우의 파일 시스템 구성 예를 보여 준다. 그림에서 보듯이 FTL은 플래시 메모리를 가상의 블록 장치로 변환해 주여 따라서 일반적인 파일 시스템이 FTL의 상단에 위치하여 동작할 수 있게 된다. 반면, 그림 1-(B)는 특별히 설계된 플래시 메모리 전용 파일 시스템의 예를 보여 준다. 이 경우에는 FTL과 파일 시스템의 구분이 없으며, 플래시 메모리 전용 파일 시스템 내부에 FTL 기능을 포함하고 있다. 이런 플래시 메모리 전용 파일 시스템의 대표적인 예로는 LFS(Log-structured File System)[2,3,4,5]의 일종인 JFFS2[4] 및 YAFFS[5] 등을 들 수 있다.

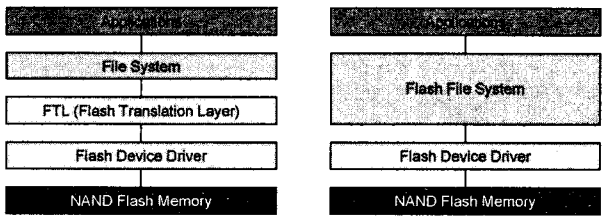

그림 1. 플래시 메모리 파일 시스템의 구성 방법

두 가지 파일 시스템 설계 방식 중에서 어떤 방식이 더 좋은지에 대해서는 여러 가지 논란이 있을 수 있다. 그렇지만 실질적인 측면을 생각한다면 그림 1-(A)와 같은 방식을 선호하게 되는데, 이는 마이크로소프트사의 FAT 파일 시스템이 휴대용 단말기 및 임베디드 시스템 분야에서는 사실상의 산업계 표준으로 사용되고 있기 때문이다. 특히, SD/MMC 등의 메모리 카드는 그 표준에 의해 FAT 구조를 갖도록 포맷되기 때문에 이러한 메모리 카드를 장착할 수 있게 설계된 시스템일 경우 반드시 운영체제 내에 FAT 파일 시스템을 갖추고 있어야 한다. 따라서 이 경우에는 내장 플래시 메모리 및 외장 메모리 카드를 위한 파일 시스템을 별도로 두지 않고, 그림 2에서 보이는 것처럼 하나의 FAT 파일 시스템으로 모두를 관리하도록 하는 것이 자원 절약 측면에서 보다 유리할 것이다.

이상과 같은 이유로 인하여 플래시 메모리가 탑재된 상용 제품에 대해서 많은 경우 FTL 및 FAT 파일 시스템의 조합으로 플래시 메모리를 관리하고 있다. 그러나 FAT 파일 시스템은 플래시 메모리의 동작 특성을 전혀 반영하지 않은 파일 시스템이기 때문에 이를 있는 그대로 시스템에 적용할 경우 성능 문제를 야기할 수 있다[6]. 특히, FAT 파일 시스템의 파일 삭제

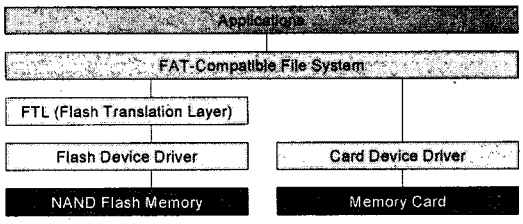

그림 2. FAT 파일 시스템의 활용 예

시 기본 동작은 플래시 메모리 및 FTL의 동작 특성을 고려하지 않은 결과로 추후에 이루어지는 파일 쓰기 동작에 대해서 적지 않은 성능 저하를 초래할 수 있다.

이에 본 논문에서는 FAT 호환 플래시 메모리 파일 시스템을 위한 성능 최적화 기법의 하나로 수정된 파일 삭제 알고리즘을 제안한다. 또한 FAT 파일 시스템의 수정된 파일 삭제 동작을 지원하기 위한 추가의 FTL 기능 및 이를 위한 API(Application Programming Interface)를 정의를 제시한다.

본 논문은 다음과 같이 구성되어 있다. 2장 연구 배경에서는 FTL의 기본 동작 원리 및 FAT 파일 시스템의 구조에 대해서 살펴 보고, 파일 삭제 동작에 대한 최적화 작업이 필요한 이유에 대해서 설명한다. 3장에서는 수정된 파일 삭제 알고리즘에 대해서 설명하고 이를 지원하기 위한 추가의 FTL 기능을 정의한다. 4장에서 제안된 파일 삭제 알고리즘이 전체 파일 시스템 성능에 미치는 영향에 대해서 실험 결과를 통해 평가한 다음, 5장에서 본 논문을 끝맺는다.

2. 연구 배경

2.1 플래시 메모리의 기본 특성

플래시 메모리는 물리적으로 블록 및 페이지로 구성된다. NAND 플래시 메모리의 경우 한 블록은 보통 16KB(소블록 장치) 또는 128KB(대블록 장치)로 구성되며, 블록 내의 페이지는 보통 512B(소블록 장치) 또는 2KB(대블록 장치)로 구성된다.

NAND 플래시 메모리에 대한 기본 동작 중, 읽기와 쓰기 동작은 페이지 단위로 일어나며 소거 동작은 블록 단위로 일어난다[7].

2.2 FTL (Flash Translation Layer)

FTL은 앞서 설명한 것처럼 플래시 메모리가 가상의 블록 장치처럼 보이도록 변환해 주는 역할을 하는 소프트웨어 모듈이다. 구체적으로 FTL은 섹터 단위로 접근되는 가상의 블록 장치를 파일 시스템에게 제공해 준다. 따라서 논리적인 주소인 섹터 번호, 즉 (논리 블록 번호, 논리 페이지 번호)를 물리적인 플래시 메모리 주소, 즉 (물리 블록 번호, 물리 페이지 번호)로 변환하기 위한 일종의 주소 사상 테이블(address mapping table)이 필요하게 된다.

이 사상 테이블을 구성하는 방법은 사상의 정밀도에 따라 블록 수준 사상(block-level mapping) 기법과 페이지 수준 사상(page-level mapping) 기법으로 나뉘는데 일반적으로 블록 수준 사상 기법이 널리 사용되고 있다[8]. 블록 수준 사상 테이블의 경우 논리 블록 번호와 물리 블록 번호 사이의 사상 관계 및 해당 블록 내에서 유효한 데이터가 저장되어 있는 페이지들에 대한 정보를 저장하게 된다.

앞서 언급하였듯이 플래시 메모리에 저장되어 있는 데이터는 그 자리에서 바로 갱신될 수 없다. 따라서 새로운 데이터는 일단 사용되지 않고 있는 임시 블록에 먼저 기록한다. 그리고 원래의 블록에 있던 유효한 데이터를 임시 블록으로 복사하여 기록한 다음, 주소 사상 테이블에 주어진 논리 주소에 해당되는 블록이 새로운 블록(임시 블록)으로 변경되었음을 기록함으로써 데이터 갱신 작업을 완료하게 된다.

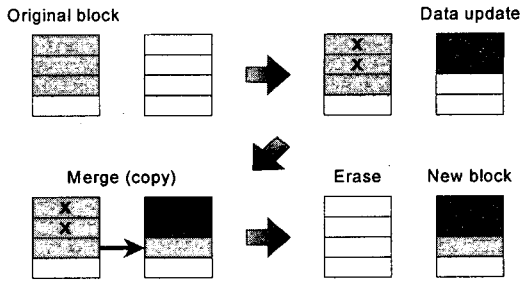

그림 3. FTL의 병합(merge) 작업

이와 같은 일련의 동작을 보통 병합(merge) 작업이라고 부르는데(그림 3 참조), 이 과정에서 여러 번의 페이지 복사와 블록 소거 등의 오버헤드가 발생할 수 있다. 따라서 이러한 병합 작업에 따르는 오버헤드를 최소한으로 줄이는 것이 FTL 알고리즘의 핵심이 된다.

2.3 FAT 파일 시스템

FAT 파일 시스템은 비교적 단순한 구조를 갖는데, 전체적으로 보면 MBR(Master Boor Record)에 해당되는 섹터 다음에 FAT(File Allocation Table) 테이블, 그리고 그 다음에 루트 디렉토리 정보를 갖는다. 디렉토리는 저장되어 있는 파일 또는 서브 디렉토리에 대한 기본 정보를 담고 있는데, 그 안에는 그 파일이 저장되어 있는 첫 번째 클러스터[1]에 대한 정보도 포함되어 있다. 해당 파일의 두 번째 클러스터부터는 FAT 테이블을 참조하여 그 위치를 파악할 수 있다.

FAT 파일 시스템에서 파일 삭제 동작은 비교적 단순하게 구현되고 있다. 즉, 어떤 파일의 삭제가 요청된 경우, FAT 파일 시스템은 먼저 주어진 파일이 위치하는 디렉토리 위치를 먼저 찾는다. 그리고 그 디렉토리 내에서 주어진 파일에 해당하는 엔트리의 첫 번째 바이트를 특별한 값으로 변경시킴으로써 해당 파일이 삭제되었음을 나타낸다. 또한 해당 파일이 차지하고 있던 클러스터들을 모두 사용되지 않은 영역으로 표기하기 위해 FAT 테이블에 있는 해당 엔트리들을 모두 클리어 한다.

그런데 이와 같은 기본적인 파일 삭제 동작이 플래시 메모리 및 FTL 위에서 이루어질 경우 그 전에는 없었던 다음과 같은 문제를 야기할 수 있다. 즉, 파일 삭제 시 디렉토리 엔트리와 FAT 엔트리들에 대해서만 조작을 할 뿐 파일 내용이 차지하고 있던 클러스터 자체에 대해서는 아무런 작업도 하지 않기 때문에, FTL은 해당 클러스터들이 여전히 사용되고 있는 것으로 간주하게 된다. 그 결과 해당 클러스터들에 포함되어 있었던 유효하지 않은 데이터들을 유효한 데이터들로 간주하게 되어 병합 작업 시 불필요한 페이지들을 복사하는 결과를 초래하게 된다. 그리고 이는 곧 파일 시스템의 파일 쓰기 성능을 저해하는 요인이 되기 때문에 개선할 필요가 있다.

3. FTL-MAP-DESTROY 기법

1) 고정된 크기를 갖는 섹터들의 집합으로 FAT 파일 시스템에서는 클러스터 단위로 빈 공간을 파일에 할당한다.

이 장에서는 FAT 파일 시스템의 기본적인 파일 삭제 동작의 문제점을 해결하기 위한 FTL-MAP-DESTROY 기법에 대해서 설명한다.

3.1 FTL-MAP-DESTROY 기법의 기본 개념

제안된 이 기법의 기본 개념은 파일 삭제 시 무효화되는 데이터 클러스터들에 대한 정보도 파일 시스템이 FTL에게 명시적으로 전달하게 하자는 것이다. 그렇게 함으로써, FTL은 주소 사상 테이블에서 무효화된 클러스터에 대한 페이지들을 더 이상 사용되지 않는 페이지들로 표기함으로써 이후 발생 가능한 병합 작업의 오버헤드를 감소시킬 수 있다.

3.1 FTL-MAP-DESTROY 기법의 구현

제안된 FTL-MAP-DESTROY 기법을 구현하기 위해서는 FTL 및 파일 시스템 모두에 대해서 다음과 같은 수정 작업을 해 주어야 한다.

FTL

FTL은 파일 시스템으로부터 무효화된 클러스터들에 대한 정보를 받기 위한 API를 파일 시스템으로 제공해 주어야 한다. 이 API에 대한 프로토타입은 다음과 같이 표현될 수 있다.

```
ftl_map_destroy(int flash_dev, int sect_no,
                int num_sects)
```

여기서 flash_dev는 해당 플래시 메모리 장치를, sect_no는 무효화되는 클러스터들의 시작 섹터 번호를, num_sects는 무효화되는 연속된 클러스터들의 총 섹터 개수를 의미한다.

FTL은 이 함수가 파일 시스템에 의해 호출되면 주소 사상 테이블을 변경하여 지정된 섹터들에 해당되는 페이지들을 더 이상 유효한 데이터를 가지고 있지 않은 페이지들로 표시한다.

파일 시스템

FAT 파일 시스템은 파일 삭제 시 파일 삭제로 인해 반환되는 클러스터들에 대해 위의 ftl_map_destroy 함수를 호출하여 해당 섹터들이 FTL 수준에서도 무효화될 수 있도록 해야 한다.

4. FTL-MAP-DESTROY 기법에 대한 성능 평가

제안된 기법은 특히 파일의 생성, 삭제 및 갱신이 빈번한 경우에 더 큰 효과를 보인다. 따라서 성능 평가에 사용되는 작업부하(workload)에 따라 그 결과가 다르게 나타날 수 있다. 즉, PDA처럼 파일 생성, 삭제, 갱신이 빈번한 경우라면 그 효과가 크겠지만, 디지털 카메라처럼 그렇지 못한 경우에는 그 효과가 작게 나타날 수 있다.

그러나 그러한 평가를 위한 실제 작업부하를 구하는 것이 어려운 관계로 본 논문에서는 가공의 작업부하를 사용하여 최대 어느 정도까지 성능 개선이 가능한지 실험해 보았다.

실험에 사용된 개발 보드는 Samsung S3C2410 MCU (ARM9 core) 기반의 보드이며 실험에 사용된 플래시 메모리는 Samsung 128MB NAND 플래시 메모리로 실제 실험에는 앞쪽의 10MB 영역을 사용하였다.

실험은 운영체제가 없는 환경 하에서 자체적으로 구현한 FTL 및 FAT 호환 파일 시스템을 사용하여 실시하였다. 실험 시나리오는 간단하게 설명하여 다음과 같다. 즉, 4KB의 동일한 크기를 갖는 파일들을 생성하여 NAND 플래시 메모리의 10MB 영역 모두를 완전히 채운 다음, 짝수 번째 파일들을 모두 삭제하였다. 그 뒤 약 4MB 크기의 파일을 기록한다. 그리고 이번에는 홀수 번째 4KB 파일들을 모두 삭제한다. 그러면 10MB 영역에는 4MB 파일 하나만 기록된 상태가 되는데, 이 상태에서 이 4MB 파일의 데이터를 모두 갱신하도록 한다.

이상과 같은 실험을 제안된 기법을 사용하는 경우와 그렇지 않은 경우로 나누어 실시하였다. 그러면 제안된 기법에 의해서 감소되는 불필요한 FTL 병합 오버헤드에 의해 어느 정도 성능 개선 효과가 있는지를 파악할 수 있다.

실험 결과 제안된 기법을 사용하지 않는 경우 4MB 파일을 갱신하는데 소요된 시간은 약 1.591 초이며 제안된 기법을 사용한 경우에는 약 1.133 초였다. 따라서 제안된 기법에 의한 파일 쓰기(갱신) 성능 개선 효과는 약 29%로 나타났다.

5. 결 론

본 논문에서는 FAT 호환 플래시 메모리 파일 시스템에 대해서 그 성능을 개선할 수 있는 최적화 기법을 제안하였다. 제안된 기법은 FAT 파일 시스템 내에 구현되어 있는 기존의 파일 삭제 동작을 확장하여, 무효화되는 데이터 클러스터들에 대한 정보도 FTL로 전달함으로써 추후 발생 가능한 FTL 병합 작업의 오버헤드를 감소시키는 효과를 가져 온다. 그 결과 플래시 파일 시스템의 전반적인 파일 쓰기 성능이 개선될 수 있다. 예비적인 실험 결과에 의하면 파일 쓰기 성능이 약 29% 정도 개선될 수 있음을 확인할 수 있었다.

향후에는 제안된 기법의 보다 객관적인 평가를 위해 휴대폰, PDA, 디지털 카메라 등으로부터 추출된 실제 작업부하를 사용하여 성능 평가 실험을 실시할 계획이다. 이를 위해 실제 작업부하를 추출하는 방법 또는 그러한 작업부하의 추출이 어려울 경우 실제 작업부하와 유사한 작업부하를 합성(synthesis)하는 방법을 찾는 것이 선행 연구과제로 필요할 것으로 판단된다.

6. 참고 문헌

[1] Understanding the Flash Translation Layer (FTL) Specification, Intel Corporation, 1998.

[2] A. Kawaguchi, S. Nishioka, and H. Motoda, "Flash- Memory Based File System", In *Proceedings of '95 Winter UXENIX Technical Conference*, 1995, pp. 155-164.

[3] M. Wu and W. Zwaenepoel, "eNvy: A Non-Volatile, Main Memory Storage System", In *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1994, pp. 86-97.

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

[5] YAFFS (Yet Another Flash File System) Specification Version 0.3, http://www.aleph1.co.kr/yaffs, 2002.

[6] S.-K. Kim, D.-H. Lee and S. L. Min, "An Efficient Cluster Allocation Scheme for NAND Flash Memory Based FAT File Systems", In *Proceedings of the 1st International Workshop on Software Support for Portable Storage*, 2005.

[7] NAND Flash Memory and SmartMedia Data Book, Samsung Electronics, Co., 2003.

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