이동 기기를 위한 플래시 메모리 파일 시스템
(Flash Memory File System for Mobile Devices)

배 영 현 † 최 종 무 †† 이 동 희 ††† 노 삼 혁 †††† 민 상 렬 †††††
(Young Hyun Bae) (Jongmoo Choi) (Donghee Lee) (Sam H. Noh) (Sang Lyul Min)

요 약 휴대 전화와 같은 소형 이동 기기에서 데이터 저장 장치로 널리 사용되는 플래시 메모리를 위한 파일 시스템은 고속의 데이터 쓰기 및 읽기 성능뿐만 아니라 소형 이동 기기의 사용 환경에 적합하도록 메모리 사용량이 적고 전원 오류 등의 상황에서도 데이터의 무결성을 보장하여야 한다. 본 논문에서는 파일 시스템 수준에서 플래시 메모리의 물리적 동작 특성과 데이터 저장 상태를 고려하여 데이터 쓰기 요청을 제어함으로써 성능을 최적화한 플래시 메모리 파일 시스템을 구현한다. 구현된 파일 시스템은 모든 작업을 트랜잭션 개념으로 처리하여 오류 상황에서도 저장 장치의 신뢰성을 보장하며 단순한 구조의 주소 사상 기법을 적용하여 메모리 사용량을 최소화한다. 그리고 실제 하드웨어 환경에서 제안된 기술을 구현하고 기존 플래시 메모리 파일 시스템과의 비교 측정을 통해 성능의 우수성을 보인다.

키워드 : 플래시 메모리, 파일 시스템, 소형 이동 기기, 내장형 시스템

Abstract File systems for flash memory that is widely used as a storage device for mobile devices should provide not only high-performance data reads and writes but also a guarantee on the data integrity even on a power failure. In this paper, we explain the design and implementation of a file system for flash memory that considers flash memory's physical characteristics and the data layout in the file system to give an optimized write performance. This file system guarantees the reliability against various system failures including a power failure by using the transaction concept in write processing. In addition, the file system minimizes the memory usage by using a simple static mapping. In the paper, we also describe the implementation of the file system and compare its performance with other existing flash memory file systems.

Key words : flash memory, file system, mobile device, embedded system

1. 서 론

최근 이동 전화기나 개인 정보 단말기(PDA)와 같은 소형 이동 기기는 다양한 멀티미디어 데이타를 처리할 수 있도록 고성능화 되고 기능이 크게 확장되고 있다. 비교적 용량이 큰 멀티미디어 데이타나 확장된 기능을 위한 다량의 정보를 원활하게 처리하기 위해서는 효율적인 데이타 저장 장치와 고성능의 파일 시스템이 필요하다. 디스크 저장 장치는 데이타의 집적도가 높아 일반적인 데이타 저장 장치로 널리 사용되고 있다. 그러나 큰 부피, 큰 소비 전력, 그리고 외부 충격에 대해 취약한 내구성 때문에 소형 이동 기기에는 적합하지 않다. 플래시 메모리는 최근 주목 받는 반도체(solid-state) 저장 장치로서 소형 이동 기기에 적합한 특성을 가지고 있다. 비휘발성 반도체 소자로서 빠른 데이타 접근 성능을 보장하며, 부피와 소비 전력이 매우 작다. 또한 외부 충격이나 자기장 등으로 인해 물리적으로 열악한 환경에서 저장된 데이타에 대한 높은 신뢰성을 보장한다.

데이타 저장 용도로 많이 사용되는 NAND 형 플래시 메모리는 다수 개의 블록(block)으로 구성된다. 하나의 블록에는 32개의 페이지(page)가 포함되며, 하나의 페이지는 512 바이트의 데이타를 저장하는 영역(main area)과 16 바이트의 부가 정보를 저장하는 영역(spare area)으로 구성된다. 따라서 하나의 블록에는 16KB의 데이타를 저장할 수 있고, 전체 블록의 개수가 4096개인 플래시 메모리 칩에는 64MB의 데이타를 저장할 수 있다.

† 학생회원 : 서울대학교 전기컴퓨터공학부
yhbae@archi.snu.ac.kr
†† 종신회원 : 단국대학교 정보컴퓨터학부 교수
choijm@dku.edu
††† 정 회 원 : 서울시립대학교 컴퓨터과학부 교수
dhlee@venus.uos.ac.kr
†††† 종신회원 : 홍익대학교 정보컴퓨터공학부 교수
samhnoh@hongik.ac.kr
††††† 종신회원 : 서울대학교 전기컴퓨터공학부 교수
symin@archi.snu.ac.kr
논문접수 : 2004년 7월 27일
심사완료 : 2005년 5월 19일

이러한 플래시 메모리는 기존의 디스크나 반도체 메모리와 다른 고유의 동작 특성을 가지고 있다. 먼저, 데이타를 기록(program)하기 위해서는 해당 메모리 영역이 미리 소거(erase)되어 있어야 한다. 또한 데이타를 기록하는 단위는 페이지인 반면에 소거는 블록 단위로 수행된다. 따라서 기존의 데이타를 갱신하여 새로 기록하고자 할 때는 동일한 블록에 속하면서 변경되지 않은 페이지들까지 복사한 후에 다시 기록해야 하는 비용이 추가로 발생한다. 그리고 블록을 소거하고 다시 데이타를 기록하는 동작이 원자적(atomic)이지 않기 때문에 중간에 오류가 발생하면 기존 데이타의 안전성도 보장하지 못한다. 이러한 제약으로 인해 플래시 메모리를 기반으로 데이타 저장 장치를 구현할 때는 별도의 소프트웨어를 이용하여 사용자에게 투명하고 안전한 데이타 읽기 및 쓰기 기능을 제공하여야 한다. 플래시 변환 계층(Flash Translation Layer, FTL)이라고 불리는 소프트웨어는 일반적으로 고유의 주소 사상(address mapping) 기법을 이용하여 플래시 메모리 상에서 디스크 저장 장치와 동일한 사용자 인터페이스를 제공한다. 즉, 갱신되는 데이타를 플래시 메모리의 새로운 영역에 기록하고 데이타의 논리적인 주소와 플래시 메모리 상의 물리적 위치 사이의 사상 관계를 갱신하여 주소를 재사상함으로써 (address remapping) 플래시 메모리의 제약을 극복할 수 있다.

플래시 메모리가 장착된 정보기기에서 플래시 변환 계층(FTL)이 제공되는 경우에는 기존의 파일 시스템을 그대로 적용하여 플래시 메모리 파일 시스템을 구현할 수 있다. 그러나 소형 이동 기기에 내장되는 파일 시스템은 전원 오류 등의 상황에서도 데이타의 안전성을 보장해야 하고 적은 시스템 자원으로도 효율적으로 동작할 수 있도록 자원 사용량을 최소화하여야 한다. 따라서 소형 이동 기기에서의 다양한 요구 사항을 만족시키는 전용의 플래시 메모리 파일 시스템이 필요하다. 또한 디스크를 위한 파일 시스템은 플래시 메모리의 동작 특성을 고려하지 않고 데이타를 처리하기 때문에 플래시 메모리에 최적화된 성능을 얻지 못하는 문제점이 있다. 일반적으로 파일 시스템은 데이타를 처리할 때 저장 매체의 물리적 특성까지 고려함으로써 성능을 향상시킬 수 있다. 예를 들어, 버클리 대학의 FFS(Fast File System)는 파일 시스템에서 논리적으로 연속된 데이타를 동일한 디스크 실린더에 할당함으로써 디스크 저장 장치에 최적화된 성능을 제공한다[1]. 플래시 메모리에서는 동작 특성에 따른 제약으로 인해 새로운 데이타의 기록보다 기존 데이타를 갱신하는 동작에서 비용이 더 많이 든다. 따라서 이러한 플래시 메모리의 동작 특성을 고려하여 데이타를 처리함으로써 성능적인 측면에서도 플래시 메모리에 최적화된 파일 시스템을 구현해야 한다.

본 논문에서는 소형 이동 기기를 위한 플래시 메모리 파일 시스템을 구현함에 있어서 파일 시스템이 갖추어야 할 요구 조건을 만족시킴과 동시에 플래시 메모리의 동작 특성을 고려하여 성능을 최적화하는 설계 기법을 제시한다. 파일 시스템은 트랜잭션(transaction) 개념으로 모든 작업을 처리함으로써 전원 오류 시에도 파일 시스템의 무결성(integrity)을 보장한다. 또한 단순한 구조의 주소 사상 기법을 기반으로 플래시 메모리를 사용함으로써 파일 시스템에서 필요로 하는 부가 정보를 감소시켜 시스템 메모리 사용량을 최소화한다. 성능적인 측면에서도 플래시 메모리에서의 데이타 갱신 동작을 효율적으로 처리할 수 있도록 주소 사상 구조와 파일 시스템의 데이타 영역 관리 기법을 설계함으로써 쓰기 성능을 개선한다. 즉, 파일 시스템의 일반적인 쓰기 요구에 대해서는 주소 사상 과정을 수행하지 않는 정적 사상(static mapping)을 적용함으로써 데이타 갱신을 위한 추가 비용 없이 고속으로 처리한다. 그리고 필요한 경우에는 원자적 쓰기(atomic write) 기능을 제공하는 별도의 주소 사상 기법을 이용하여 처리함으로써 데이타의 안전성은 언제나 보장한다. 이와 같이 구현된 플래시 메모리 파일 시스템은 소형 이동 기기에 보다 적합한 성능 특성을 보인다. 또한 파일 시스템의 활용률(utilization)이나 데이타 저장 상태에 관계없이 일관성 있는 성능 특성을 보인다. 본 논문에서는 플래시 메모리를 장착한 개발 보드에서 실제로 비교 측정한 결과를 통해 이러한 성능 특성을 확인할 수 있다.

본 논문은 다음과 같이 구성된다. 2장에서는 기존의 플래시 메모리 파일 시스템 연구와 구현을 살펴본다. 3장에서는 소형 이동 기기를 위해 구현한 플래시 메모리 파일 시스템을 설명한다. 소형 이동 기기에서의 요구 조건을 충족시키는 파일 시스템 설계 기법 및 플래시 메모리의 동작 특성을 고려한 성능 최적화 기법을 제시한다. 4장에서는 실제 하드웨어 개발 환경을 통해 구현된 플래시 메모리 파일 시스템의 성능을 측정하고 기존 플래시 메모리 파일 시스템과 비교 분석한 결과를 제시한다. 마지막으로 5장에서 본 논문의 결과를 정리하고 향후 과제를 제시하며 결론짓는다.

2. 기존 연구 및 구현

플래시 메모리는 기존의 하드 디스크나 반도체 메모리와 다른 고유의 동작 특성을 가지고 있다. 특히, 저장된 데이타를 직접 갱신(in-place update)할 수 없는 제약이 있기 때문에 플래시 메모리를 이용하여 데이타 저장 장치를 구현하기 위해서는 특별한 소프트웨어적 고려가 필요하다. 이러한 제약을 극복하고 효율적인 저장

장치를 구현하기 위한 연구가 있어왔다[2,3].

플래시 메모리 파일 시스템은 플래시 메모리를 기반으로 하여 사용자에게 보다 효율적인 저장 장치 환경을 제공한다. 하드 디스크와 동일한 저장 장치 환경을 제공하는 플래시 변환 계층(FTL)의 접목 형태에 따라 플래시 메모리 파일 시스템의 구현은 크게 두 가지 방식으로 구분된다. 먼저, 독립된 플래시 변환 계층(FTL)을 구현하고 기존 파일 시스템을 큰 변형 없이 그대로 적용하는 방식이 있다. 메모리 카드를 이용하는 정보기기에서는 메모리 카드에 내장된 플래시 변환 계층이 제공하는 데이타 입출력 기능을 이용하여 기존 파일 시스템을 그대로 사용할 수 있다. 또한 M-Systems의 TrueFFS는 플래시 메모리가 내장된 정보기기에서 플래시 변환 계층 역할을 수행하여 기존 파일 시스템이 플래시 메모리에 대한 고려 없이 데이타 저장 기능을 구현할 수 있도록 한다[4]. 이와 같이 독립된 플래시 변환 계층을 이용하는 경우에 파일 시스템의 구현은 비교적 쉽게 이루어지나 데이타 입출력 요청을 생성하는 파일 시스템에서 플래시 메모리의 특성을 반영할 기회가 제한되기 때문에 보다 최적화된 플래시 메모리 파일 시스템을 구현하기가 어렵다. 반면에 파일 시스템에서 플래시 변환 계층의 역할까지 통합하여 구현하는 방식은 파일 시스템의 설계에 플래시 메모리의 동작 특성을 보다 적극적으로 반영할 수 있다. JFFS2(Journaling Flash File System 2)와 YAFFS(Yet Another Flash File System)는 리눅스(LINUX) 운영체제에서 동작하는 대표적인 플래시 메모리 파일 시스템으로서 후자의 구현 방식을 채택하고 있다.

JFFS2는 로그 구조 파일 시스템(log-structured file system)의 원리를 플래시 메모리 파일 시스템의 설계에 적용하여 구현한 예이다[5]. 로그 구조 파일 시스템은 저장 장치의 새로운 영역에 변경된 데이타를 계속 추가해 나가는 방식으로 파일 시스템의 데이타 갱신 작업을 구현하였다[6]. 이것은 데이타의 갱신 동작에서 많은 비용이 드는 플래시 메모리를 위한 효과적인 접근 방법이라고 할 수 있다. 그러나 JFFS2는 전체 플래시 메모리 블록이 한번 사용된 후에는 자유 블록을 확보하는 비용이 과다하게 많이 들고 파일 시스템을 구동하기 위한 메모리 사용량도 과도하게 크다. 또한 파일 시스템이 새로 시작될 때 전체 저장 공간을 검색하여 사상 정보를 재구축하기 때문에 초기 구동 시간이 길다. 저널링(journaling) 기법을 이용하여 구현된 YAFFS는 JFFS2에 비해 개선된 성능을 제공한다[7]. 그러나 YAFFS도 새로운 자유 블록을 확보하기 위한 쓰레기 수집(garbage collection) 비용으로 인해 성능 개선의 한계를 가지고 있다. 또한 플래시 메모리의 페이지마다 부가 정보를 유지해야 하기 때문에 파일 시스템의 용량에 비례하여 시스템 메모리 사용량이 비교적 크게 증가하는 문제가 있다. 따라서 플래시 메모리의 동작 특성을 보다 심층적으로 고려하여 설계함으로써 소형 이동 기기에 적합한 성능 특성을 제공하면서도 성능이 뛰어난 플래시 메모리 파일 시스템을 구현하기 위한 연구가 필요하다.

3. 플래시 메모리 파일 시스템의 설계 및 구현

플래시 메모리 파일 시스템은 고속의 데이타 입출력 성능뿐만 아니라 시스템 오류에 대해서도 저장된 데이타의 안전성을 보장할 수 있어야 한다. 특히, 소형 이동 기기에서의 효율적인 구현을 위해서는 메모리 등의 자원 사용량이 매우 적어야 한다. 본 논문에서 구현한 트랜잭션 기반의 플래시 메모리 파일 시스템(Transactional Flash Memory File System, TFFS)은 플래시 메모리의 동작 특성을 고려한 설계를 통해 이러한 요구 조건을 만족시키고 있다. TFFS에서 파일의 생성 등과 같이 파일 시스템의 내용이 변경되는 모든 작업은 트랜잭션 개념에 따라 처리된다. 한 트랜잭션의 시작부터 종료까지는 논리적으로 원자적이기 때문에 중간에 오류가 발생하더라도 파일 시스템은 트랜잭션 시작 직전의 상태로 안전하게 복구된다. 따라서 파일 시스템은 이동 정보기기에서 발생할 수 있는 다양한 오류 상황에서 언제나 무결성(integrity)을 보장할 수 있다. TFFS에서의 트랜잭션 처리는 교대 사상(alternate mapping)이라고 하는 특별한 주소 사상 기법을 통해서 구현된다. 교대 사상 기법은 기존 데이타를 안전하게 유지하면서 플래시 메모리의 교대 영역에 데이타를 기록하고 마지막으로 변경된 주소 사상 정보를 한번에 기록한다. 이에 따라 전체 데이타 기록 동작이 원자적으로 수행되고 이것을 통해 파일 시스템 트랜잭션의 원자성(atomicity)을 보장하는 것이다. 또한 교대 사상은 트랜잭션 내에서 원자적 기록이 필요한 일부 데이타에 대해서만 수행되기 때문에 사상 정보의 크기가 매우 작고 플래시 메모리에 갱신하고 저장하는 절차가 단순하다.

플래시 메모리에 저장되어 있는 데이타를 원자적으로 안전하게 갱신하기 위해서는 이와 같이 새로운 영역에 변경된 데이타를 기록하고 주소 사상 정보를 갱신하는 주소 재사상이 필수적이다. 그런데 기존의 데이타가 유효하지 않은 경우에는 해당 영역을 직접 소거하고 변경된 데이타를 기록하더라도 저장된 데이타의 안전성에는 아무런 문제가 없다. 이때에는 주소 재사상을 위해 플래시 메모리에서 자유 블록을 확보하는 비용이나 사상 정보를 갱신하는 비용 없이 보다 빠른 속도로 데이타 기록 작업을 처리할 수 있다. 이러한 플래시 메모리의 동작 특성은 플래시 메모리 파일 시스템의 성능 개선에

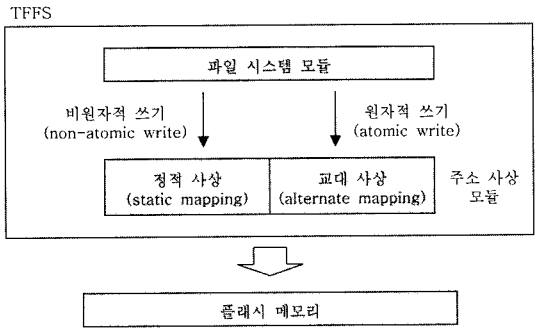

그림 1 TFFS의 구조

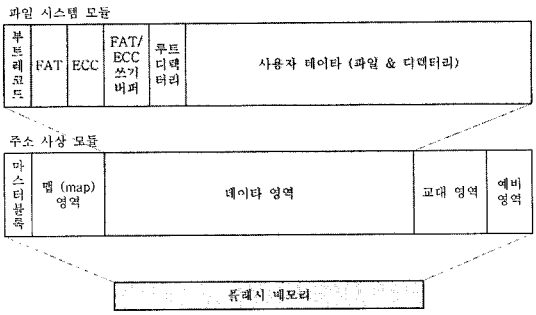

그림 2 TFFS의 영역 할당

중요한 동기를 제공한다. 즉, 파일 시스템에서는 클러스터(cluster)의 집합으로 표현되는 모든 데이타 영역에 대해 저장된 데이타의 유효성을 쉽게 알 수 있다. 따라서 TFFS에서는 새로 데이타가 쓰여질 영역의 상태에 따라 주소 재사상 과정을 선택적으로 적용함으로써 파일 시스템의 쓰기 성능을 개선한다. 또한 유효한 데이타가 저장된 영역을 갱신하는 경우에도 클러스터 교체(cluster swapping) 기법을 적용하여 유효하지 않은 영역에 새로 갱신된 데이타를 기록함으로써 성능 개선의 폭을 확대한다. 이와 같이 TFFS는 플래시 메모리의 동작 특성을 고려하여 성능을 최적화하였고 트랜잭션 개념에 따른 설계를 통해 소형 이동 기기에서 안정적인 파일 시스템 기능을 제공한다. 다음 절들에서 TFFS의 구조 및 이원화된 주소 사상 기법, 트랜잭션 처리 과정, 그리고 성능 최적화 기법 등에 대해 구체적으로 설명한다.

3.1 파일 시스템의 구조

본 논문에서 구현한 플래시 메모리 파일 시스템(TFFS)은 그림 1에서 보는 것처럼 파일 시스템 모듈과 주소 사상 모듈로 구성된다. 파일 시스템 모듈은 파일 및 디렉터리의 생성, 쓰기, 갱신, 읽기 등의 기능을 사용자에게 제공하며 주소 사상 모듈을 통해 플래시 메모리에 접근한다. 주소 사상 모듈은 플래시 변환 계층(FTL)과 유사한 기능을 제공하지만, 파일 시스템의 지능적인 데이타 쓰기 정책에 따라 선택적으로 주소 사상 기법을 적용하여 플래시 메모리에 데이타를 기록한다. 파일의 확장을 위하여 데이타를 쓰는 경우에는 중간에 오류가 발생하더라도 기존 데이타를 복구할 필요가 없기 때문에 데이타 쓰기를 비원자적으로 처리할 수 있다. 이때는 보다 고속으로 데이타 쓰기를 처리할 수 있도록 정적 사상에 따라 할당된 플래시 메모리의 물리적인 위치를 직접 소거하고 데이타를 기록한다. 반면에 파일이나 디렉터리가 갱신되는 경우 혹은 파일 시스템의 부가 정보(meta-information)를 기록하는 경우에는 동작이 완료될 때까지 반드시 기존 데이타를 유지하여 오류가 발생한 후에도 안전하게 복구할 수 있어야 한다. 이때는 교대 사상이 제공하는 원자적 쓰기 기능을 이용하여 데이타를 기록한다. 이와 같이 이원화된 주소 사상 구조는 플래시 메모리의 동작 특성을 고려한 설계로서 파일 시스템 모듈에서의 지능적인 제어를 통해 쓰기 성능을 크게 향상시킬 수 있고 동시에 트랜잭션의 원자성을 보장할 수 있다. 한편, TFFS에서 파일을 읽는 경우에도 주소 사상 모듈을 통하여 플래시 메모리에 저장된 데이타를 읽는다. 이때는 먼저 사상 정보를 검색하여 교대 사상을 통해 기록된 데이타인지를 식별하고 그렇지 않은 경우에는 정적 사상에서 고정된 위치의 데이타를 읽는다.

TFFS의 파일 시스템 모듈과 주소 사상 모듈은 그림 2에서 보는 것처럼 각각 독자적인 영역 할당 정책에 따라 플래시 메모리의 저장 공간을 사용한다. 주소 사상 모듈은 전체 플래시 메모리에 다섯 개의 영역을 블록 단위로 할당하여 파일 시스템 모듈에서 요청하는 데이타 접근에 대해 주소 사상 기능을 수행한다. 마스터 블록(master block)은 플래시 메모리의 첫 번째 물리 블록에 할당되며 주소 사상 모듈의 영역 할당 정보가 저장된다. 데이타 영역(data area)은 파일 시스템 모듈의 논리적인 주소 공간이 정적 사상에 따라 할당된 플래시 메모리의 물리 블록들로 구성된다. 맵 영역(map area)과 교대 영역(alternate area)은 교대 사상을 처리하기 위한 공간이다. 원자적으로 처리되어야 하는 쓰기 요청에 대해서 데이타 영역의 내용은 그대로 유지한 채 교대 영역에 새로운 데이타를 기록한다. 그리고 변경된 교대 영역의 사상 정보를 맵 영역에 기록함으로써 원자적인 데이타 갱신을 보장한다. 마지막으로 예비 영역(reserved area)은 파일 시스템의 사용 중에 데이타, 맵, 교대 영역에서 발생할 수 있는 불량 블록(bad block)을 교체할 수 있도록 파일 시스템이 초기화될 때 미리 확보해두는 물리 블록들이다.

파일 시스템 모듈의 주소 공간은 섹터(sector)의 집합으로 이루어진 클러스터(cluster) 단위로 할당된다. 여기

서 하나의 섹터는 512 바이트로서 플래시 메모리의 한 페이지에 저장되는 크기이다. 그리고 TFFS에서는 단순하면서도 플래시 메모리에 최적화된 성능을 구현하기 위해 클러스터의 크기를 플래시 메모리의 물리 블록 크기와 동일하게 설정한다. 그림에서 보는 것처럼 파일 시스템 모듈은 주소 사상 모듈이 제공하는 데이타 영역 위에서 구성된다. 클러스터 할당 구조는 기존의 FAT (File Allocation Table) 파일 시스템과 유사하며 ECC 영역과 쓰기 버퍼(write buffer) 영역이 추가되었다. 첫 번째 클러스터의 첫 번째 섹터는 부트 레코드(boot record)로 사용되며 파일 시스템 모듈의 영역 할당 정보 등이 저장된다. FAT 영역에는 파일 시스템 모듈의 전체 클러스터 사용 정보가 저장된다. 데이타가 저장되어 사용되고 있는 클러스터들에 대해 동일한 파일 별로 연결 고리(클러스터 체인, cluster chain)를 형성하여 파일의 할당 정보를 구성한다. ECC 영역은 파일 시스템 수준에서 데이타의 오류 정정 기능을 처리하기 위하여 사용된다. TFFS는 FAT이나 ECC 정보가 교대 사상을 통해 빈번하게 갱신되는 부담을 줄여서 성능을 개선하기 위해 FAT/ECC 쓰기 버퍼를 사용한다. 즉, FAT이나 ECC 정보가 변경될 때 전체 클러스터가 아니라 변경된 섹터만을 쓰기 버퍼에 저장한다. 또한 쓰기 버퍼 내에서는 데이타 기록을 주소 재사상 과정 없이 처리하기 때문에 전체적인 데이타 쓰기 성능을 향상시킬 수 있다. 마지막으로 FAT/ECC 쓰기 버퍼 이후의 클러스터들은 루트 디렉터리부터 시작하여 사용자 파일이나 디렉터리 데이타를 저장하기 위해 할당된다.

플래시 메모리에서는 일반적으로 각 페이지의 부가 영역(spare area)을 이용하여 데이타의 오류 정정 기능을 수행한다. 하나의 페이지에 512 바이트 데이타가 기록될 때 ECC 정보를 해당 페이지의 부가 영역에 같이 기록하고 데이타를 읽을 때 참조함으로써 비트 (bit) 단위의 오류 발생을 감지하거나 수정할 수 있다. 그런데 이러한 기술이 특허에 의해 보호되고 있어서 자유롭게 사용할 수 없기 때문에 TFFS에서는 파일 시스템 수준에서 데이타의 오류 정정 기능을 제공한다. 이를 위해 데이타 클러스터에 속한 모든 섹터마다 3 바이트씩의 ECC 정보를 생성하여 별도의 ECC 영역에 저장하고 해당 섹터를 읽을 때 참조한다. 데이타 섹터의 내용이 변경될 때에는 ECC 값을 새로 계산하여 ECC 영역에서 해당하는 섹터도 같이 변경되도록 한다. 이때 데이타 섹터와 ECC 섹터의 변경은 동일한 트랜잭션을 통해 처리함으로써 항상 일관성 있는 정보를 유지한다.

3.2 주소 사상 기법

기존 데이타의 직접 갱신이 어려운 플래시 메모리 상에서 파일 시스템을 구현할 때에는 안전하고 효율적인 갱신을 위해 고유의 주소 사상 기법을 사용하는 것이 일반적이다. 즉, 파일 시스템의 데이타가 갱신될 때 플래시 메모리 상에서 실제로 저장되는 위치를 동적으로 변화시키면서 플래시 메모리의 동작 특성에 따른 제약을 극복하는 것이다. 이때 데이타의 논리적인 주소와 플래시 메모리 상의 물리적인 주소 사이의 사상(mapping) 정보를 주소 재사상 과정에 따라 유지해야 한다. 그러나 주소 재사상 과정에 따라 데이타를 갱신하기 위해서는 새로운 플래시 메모리 영역을 확보해야 하고 이후에 사상 정보를 갱신해야 하는 비용을 감수해야 한다. TFFS에서는 이원화된 주소 사상 구조를 구현함으로써 지능적인 데이타 쓰기 관리에 따라 데이타의 안전한 갱신을 보장하면서도 주소 재사상 비용 없이 고속으로 데이타 쓰기를 처리할 수 있다. 또한 물리 블록의 크기와 동일하게 클러스터를 정의하고 별도의 교대 영역을 사용하기 때문에 파일 시스템의 활용률이나 데이타 저장 상태와 관계없이 일관성 있는 성능을 보장한다.

3.2.1 주소 사상 기법의 설계

TFFS의 주소 사상은 기본적으로 파일 시스템의 주소 공간을 플래시 메모리의 고정된 위치에 정적으로 할당한다. 주소 사상 모듈의 데이타 영역은 플래시 메모리의 연속된 물리 블록에 할당되며, 파일 시스템의 논리 블록 즉, 클러스터들이 데이타 영역 내에서 연속적으로 사상된다. 정적 사상을 통해 데이타를 쓰거나 읽는 경우에는 부가적인 사상 정보 없이 플래시 메모리에 접근할 수 있다. 또한 파일 시스템에서 사용하고 있지 않은 클러스터에 새로운 데이타를 쓰고자 할 때는 정적으로 사상되는 물리 블록을 직접 소거하고 기록할 수 있다. 이것은 해당 클러스터가 사상된 물리 블록에는 유효하지 않은 데이타가 저장되어 있기 때문이다. 한편, 사용 중인 클러스터의 기존 데이타를 갱신할 때에는 도중에 오류가 발생하더라도 파일 시스템을 안전하게 복구할 수 있도록 새로운 데이타를 원자적으로 기록해야 한다. TFFS는 교대 사상을 통해 원자적 쓰기 기능을 제공한다. 데이타 영역과는 별도로 교대 영역을 할당하고 원자적으로 갱신되는 클러스터의 데이타를 기록한다. 그리고 교대 영역에 저장된 클러스터들의 주소 사상 정보를 맵 영역에 기록한다.

그림 3의 섹터 쓰기 알고리즘은 TFFS의 파일 시스템 모듈에서 데이타 특성에 따라 각각의 사상 방식을 통해 해당 함수를 호출했을 때 주소 사상 모듈에서 처리되는 과정을 보여준다. 파일 시스템의 동일한 트랜잭션 내에서 동일한 클러스터로 요청되는 쓰기에 대해서는 3, 14 번째 줄의 물리 블록 소거 작업을 한번만 수행한다. 또한 16, 17, 18 번째 줄의 교대 사상 부가 작업도 트랜잭션이 종료될 때 한 번에 모아서 수행한다.

```
1   write_sectors_via_static_mapping(sector_number, sector_count, data) {
2       요청된 섹터들이 속한 클러스터가 사상된 물리 블록 번호를 계산
3       지정된 물리 블록을 소거 (erase)
4       요청된 개수의 섹터들을 소거된 블록의 해당 페이지에 기록 (program)
5   }

6   write_sectors_via_alternate_mapping(sector_number, sector_count, data) {
7       요청된 섹터들이 속한 클러스터의 기존 데이타가 교대 영역에 존재하는 지 확인
        : 맵 페이지의 정보에서 해당 클러스터 번호가 존재하는 지 확인
8       if (기존 데이타가 교대 영역에 존재) {
9           데이타 영역에서 해당 클러스터가 사상된 물리 블록 번호를 계산
10      } else {  // 기존 데이타가 데이타 영역에 존재
12          교대 영역에서 사용되지 않은 물리 블록을 할당
            : 모든 블록이 사용 중일 때에는 한 블록을 선택하여 모든 페이지를 데이타
              영역으로 복사한 후에 해당 블록을 확보
13      }
14      지정된 물리 블록을 소거 (erase)
15      요청된 개수의 섹터들을 소거된 블록의 해당 페이지에 기록 (program)
16      기록되지 않은 페이지들에 기존 데이타를 복사 (read & program)
17      새로 기록된 물리 블록을 위한 교대 영역의 블록 주소 사상 정보를 갱신
        : 교대 영역에 기록된 경우에는 해당 클러스터 번호로, 데이타 영역에 기록된
          경우에는 -1로 갱신
18      갱신된 블록 주소 사상 정보를 맵 영역의 한 페이지에 기록 (program)
19  }
```

그림 3 TFFS의 주소 사상 기법에 따른 섹터 쓰기 알고리즘

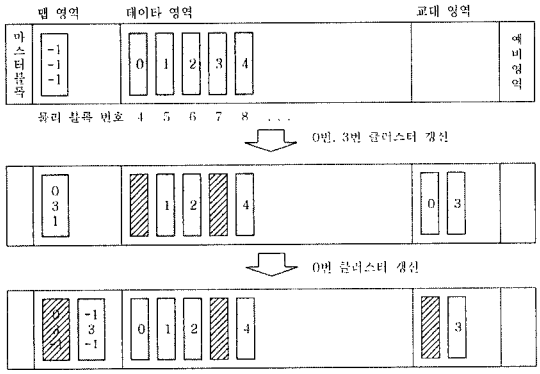

그림 4 교대 사상의 쓰기 예

즉, TFFS의 실제 구현에서는 성능 개선을 위해 주소 사상 모듈의 쓰기 처리는 파일 시스템의 트랜잭션 처리에 통합되어 수행된다.

그림 4는 TFFS의 주소 사상 모듈에서 교대 사상을 통해 몇 개의 클러스터를 갱신하는 과정을 예로 보여준다. 초기에 클러스터 0번부터 4번까지 다섯 개 클러스터가 정적 사상에 따라 데이타 영역에 저장되어 있다. 첫 번째 트랜잭션에서 파일 시스템이 0번과 3번 클러스터에 포함된 섹터들을 갱신하며, 두 번째 트랜잭션에서 0번 클러스터에 포함된 섹터들을 다시 갱신한다. 그림 4는 이러한 쓰기 요청이 처리됨에 따라 데이타 영역,. 교대 영역, 그리고 맵 영역의 맵 페이지 내용이 변경되는 과정을 보여준다.

3.2.2 주소 사상 기법의 설계 시 고려사항

교대 사상을 통해 데이타를 기록할 때 교대 영역에서 블록을 할당할 수 없다면 인위적으로 하나의 클러스터를 데이타 영역으로 옮겨야 한다. 교대 영역에 할당된 물리 블록의 개수가 많을수록 이러한 추가 비용이 발생할 가능성이 감소하기 때문에 주소 사상 모듈의 성능은 개선된다. 그러나 교대 영역의 크기가 커짐에 따라 성능이 비례하여 지속적으로 증가하지 않으며, 실제 구현을 통해 관찰해 보면 비교적 작은 개수의 물리 블록으로도 교대 사상은 효율적으로 동작한다. 예를 들어, 블록의 개수가 8192개인 128MB 플래시 메모리에서도 교대 영역에 16개의 블록만을 할당하여 구현할 수 있다. 따라서 TFFS에서는 전체 플래시 메모리에서 파일 시스템이 데이타 저장 공간으로 사용하지 못하는 블록의 비율이 매우 작다. 또한 교대 영역의 한 블록 당 2 바이트가 필요한 주소 사상 정보를 512 바이트 내로 제한할 수 있기 때문에 한번의 물리적인 페이지 기록으로 갱신된 사상 정보를 기록할 수 있다. 이것은·교대 사상을 통한 데이타 갱신의 원자성을 보장하는 것이다. 결국 TFFS는 파일 시스템의 크기가 커지더라도 교대 사상의 효율성은 보장하면서 필요한 주소 사상 정보의 크기는 최소화함으로써 시스템 자원이 제한적인 소형 이동 기기에서 보다 효율적으로 동작할 수 있다.

교대 사상에서 사상의 단위는 블록(block)이다. 따라서 한 논리 블록(클러스터)의 일부 데이타가 교대 사상을 통해 갱신될 때 블록 내에서 유효한 데이타를 저장하고 있는 나머지 페이지(섹터)들도 교대되는 블록으로 복사되어야 한다. 이것은 파일 시스템의 성능이 저하되는 원인이 된다. 그러나 TFFS의 파일 시스템 모듈에서는 몇 가지 성능 최적화 기법을 통해 교대 사상을 필요로 하는 데이타 쓰기를 최소화하여 성능 저하를 보완한다. 또한 소형 이동 기기에서 파일 시스템의 성능에 민감한 멀티미디어 데이타는 대부분 하나의 클러스터 이상으로 용량이 크며, 작은 크기의 데이타는 일반적으로 사용자의 직접적인 인터페이스를 필요로 하는 응용에 사용된다. 따라서 TFFS의 교대 사상에서 작은 크기의 쓰기에 취약한 성능 특성이 전체적인 파일 시스템의 성능이나 사용자의 체감 성능에 미치는 영향은 크지 않을 것으로 예상한다. 그리고 이러한 성능 특성은 클러스터의 크기를 보다 작게 설계하고 현재의 교대 사상 기법에 일종의 쓰기 버퍼 기능을 추가함으로써 근본적으로 보완할 수 있다.

TFFS의 주소 사상 모듈에는 NAND 형 플래시 메모리에서 발생할 수 있는 불량 블록(bad block)을 효과적으로 처리하는 기술이 포함되어 있다. 플래시 메모리의

제조 시에 발생한 불량 블록은 칩의 생산 과정에서 미리 기록해 둔 정보를 읽어서 식별할 수 있다. 또한 사용 중에는 소거 동작 혹은 기록 동작의 수행 결과에 따라 실패가 반복될 경우 불량 블록으로 판별한다. 이와 같이 식별된 불량 블록에 대한 접근 요구는 일반적으로 별도의 정상적인 블록에 대한 접근 요구로 전환시켜 처리한다. TFFS의 주소 사상 모듈은 이러한 불량 블록의 교체 정보를 마스터 블록(master block) 내에 테이블 형태로 저장하고 참조함으로써 불량 블록이 존재하는 플래시 메모리 상에서도 안정적으로 동작한다. 제조 시에 발생한 초기 불량 블록들은 파일 시스템이 초기화될 때 주소 사상 모듈에서 식별하여 불량 블록 테이블에 등록한다. 또한 파일 시스템의 사용 중에 불량 블록으로 판별되면 초기화 과정에서 미리 확보해둔 예비 영역(reserved area)의 정상 블록을 이용하여 유효한 데이타를 복사한 후에 불량 블록 테이블을 갱신한다. 따라서 플래시 메모리에 대한 모든 접근은 불량 블록 테이블을 우선 검색하여 실제로 사용되고 있는 물리 블록 번호를 식별한 후에 진행된다.

TFFS에서 고유의 주소 사상 기법에 따라 플래시 메모리를 사용할 때 할당된 영역에 따라 각 물리 블록에 대한 접근 형태가 다르기 때문에 기록 및 소거 횟수가 불균형을 이루게 된다. 즉, 맵 영역(map area)과 교대 영역(alternate area)에 속한 물리 블록들은 다른 영역에 속한 물리 블록들 보다 상대적으로 소거 횟수가 많아지게 된다. 이때 전체 플래시 메모리의 수명과 상관없이 과도하게 소거가 많이 이루어진 영역에서 불량 블록이 많이 발생하여 플래시 메모리를 더 이상 사용할 수 없게 되는 문제가 발생한다. 따라서 TFFS의 주소 사상 모듈에서는 영역 이동(area shift) 기법을 통해 전체 블록들의 소거 횟수를 균일하게 유지한다(wear-leveling). 이것은 소거 빈도가 서로 다른 맵 영역, 교대 영역, 그리고 데이타 영역의 위치를 일정한 주기와 조건에 따라 순환 이동시킴으로써 특정한 물리 블록들이 집중적으로 사용되는 것을 방지한다. 이에 따라 정적 사상에 의해 고정적이어야 하는 논리 블록의 주소 사상 관계는 변하게 된다. 그러나 이것은 주소 재사상에 의해 임의로 변한 것이 아니고 블록들의 상대적인 위치를 유지한 채로 이동한 것이기 때문에 영역 이동된 크기를 참조하여 이전 주소 사상 정보를 그대로 적용할 수 있다. 즉, 영역 이동 기법은 이동된 영역 정보를 별도의 플래시 메모리 영역에 기록하고 그 정보를 바탕으로 플래시 메모리에 대한 접근을 투명하게 처리하기 때문에 TFFS의 주소 사상 과정에는 영향을 주지 않는다.

3.3 트랜잭션 처리

파일 시스템의 내용이 변경되는 작업을 수행하는 동안 오류가 발생하면 일반적으로 파일 시스템의 전체 영역을 검색하여 불안정한 상태로 종료된 파일 시스템을 복구해야 한다. 이러한 복구 절차는 시간이 많이 걸릴 뿐만 아니라 경우에 따라서는 복구가 불가능하고 기존 데이타까지 손실되는 문제가 있다. TFFS에서는 트랜잭션(transaction) 개념으로 파일 시스템 작업을 처리함으로써 비정상적으로 종료한 후에도 별도의 복구 절차 없이 무결한 상태를 유지할 수 있다. 즉, 트랜잭션 처리의 원자적 특성(atomicity)에 따라 최종적으로 트랜잭션의 종료(commit) 절차가 수행되기 전에는 미완성된 작업의 내용과 상관없이 트랜잭션 시작 이전의 무결한 상태를 보장하는 것이다. 따라서 트랜잭션을 수행하는 중에 오류가 발생하더라도 별도의 복구 절차가 필요 없기 때문에 안전하고 신속한 재기동이 가능하다.

TFFS에서 트랜잭션은 일련의 플래시 메모리 읽기와 기록 동작으로 구성되며 주소 사상 모듈이 제공하는 트랜잭션 제어 함수들에 의해 구현된다. 각 트랜잭션은 begin_transaction() 함수를 통해 해당 트랜잭션 내에서 원자적 쓰기가 필요한 만큼의 블록을 교대 영역에서 확보하는 것으로 시작한다. 최악의 경우에 교대 사상으로 기록되어야 하는 데이타가 모두 데이타 영역에 존재한다면 새로 기록되는 데이타는 모두 교대 영역에 저장되기 때문이다. 트랜잭션의 내용으로서 수행되는 쓰기 작업은 기록될 데이타의 특성에 따라 원자적 혹은 비원자적으로 처리된다. 기존 데이타를 갱신하는 경우에는 교대 사상을 통해 원자적으로 기록하며 그렇지 않은 경우에는 정적 사상에 따라 데이타 영역에 직접 기록한다. 마지막으로 트랜잭션의 종료는 end_transaction() 함수를 호출하여 수행한다. 이 함수에서는 아직 기록되지 않은 파일 시스템의 버퍼에 대한 쓰기를 완료하고, 교대 영역의 변경된 주소 사상 정보를 맵 영역에 기록한다. 맵 영역에 대한 기록은 한번의 페이지 기록으로 완료되기 때문에 전체 트랜잭션의 원자성이 보장된다. 즉, 파일 시스템의 새로운 데이타가 플래시 메모리에 이미 기록되었지만 변경된 주소 사상 정보가 맵 영역에 기록되기 전에는 트랜잭션이 시작되기 전과 동일한 파일 시스템 상태를 보장하는 것이다.

TFFS의 주소 사상 모듈은 교대 영역의 크기 한도 내에서 원자적 쓰기를 보장한다. 따라서 파일이나 디렉터리의 삭제 동작에서는 갱신되는 데이타의 크기를 예측하기 어렵기 때문에 하나의 트랜잭션으로 처리하기 어려운 경우가 있다. 파일을 삭제할 때는 파일에 할당되었던 모든 클러스터를 반환하고 FAT 정보를 갱신해야 한다. 이때 파일의 크기가 클수록 FAT에서 갱신될 전체 섹터들을 예측하기 어렵다. 최악의 경우에는 모든 FAT 섹터가 갱신될 수 있기 때문에 교대 영역 등의

시스템 자원이 고갈되면 삭제 작업이 완료되기 전에도 트랜잭션을 종료해야만 한다. 따라서 트랜잭션이 종료된 후에도 파일 시스템이 불안전한 상태가 될 수 있는 것이다. 이와 같이 갱신되는 섹터들을 예측하기 어려운 경우에는 두 개 이상의 트랜잭션이 연속적으로 수행되어야 하고 추가의 오류 복구 기법을 이용하여 트랜잭션 처리를 보완하여야 한다. TFFS에서는 이 목적으로 주소 사상 모듈의 맵 영역에 파일 삭제와 관련된 클러스터 정보를 추가로 기록한다. 이에 따라 파일을 삭제할 때는 해당 파일에 할당된 클러스터 체인의 시작 번호를 맵 영역에 먼저 기록한 후에 클러스터를 체인의 뒤에서부터 반환하여 FAT 정보를 갱신하기 시작한다. 이러한 반환 작업 도중에 오류가 발생하면 파일 시스템은 재기동 후 맵 영역에 기록된 클러스터 번호를 참조하여 중단된 삭제 작업을 안전하게 완료할 수 있다. 즉, 트랜잭션의 종료 절차가 수행되지 않은 채 오류가 발생한 경우에는 삭제 작업을 처음부터 다시 수행할 수 있다. 또한 삭제 작업을 구성하는 일부 트랜잭션이 종료된 후에 오류가 발생하더라도 전체 파일 시스템을 검색할 필요 없이 종료된 트랜잭션 정보와 이전에 기록한 클러스터 번호로부터 클러스터 반환이 중단된 지점을 찾아 반환 작업을 다시 시작할 수 있다. 따라서 TFFS는 갱신되는 데이타의 크기가 커서 하나의 트랜잭션으로 처리할 수 없는 경우에도 오류 발생 후의 안전하고 신속한 복구를 보장할 수 있다.

3.4 성능 최적화

TFFS의 주소 사상 모듈은 정적 사상과 교대 사상이 결합된 이원적 주소 사상 구조를 제공함으로써 파일 시스템의 성능을 플래시 메모리의 동작 특성에 최적화되도록 개선할 수 있다. 즉, 파일 시스템에서 현재 유효하지 않은 영역에 데이타를 쓰는 경우에 주소 재사상에 따르는 비용 없이 고속으로 쓰기 작업을 처리할 수 있는 것이다. 따라서 TFFS의 파일 시스템 모듈에서는 쓰여질 데이타의 특성을 고려하여 쓰기 작업을 각각 다른 사상 구조에 따라 처리해야 한다. 또한 가능한 한 많은 쓰기 작업이 교대 사상을 거치지 않고 처리될 수 있도록 파일 시스템의 영역을 관리하는 최적화 기법이 필요하다.

파일 시스템의 데이타 쓰기 동작은 파일의 확장(append)과 갱신(update)으로 대표될 수 있다. 파일이 확장되는 경우에는 파일 시스템의 새로운 영역에 데이타를 기록하기 때문에 정적 사상에 따라 지정된 물리 블록을 직접 소거하고 데이타를 기록할 수 있다. 이때는 작업 도중에 오류가 발생하더라도 기존의 유효한 파일 시스템 내용은 손상되지 않으므로 파일 시스템의 무결성은 그대로 유지된다. 그러나 파일이 갱신되는 경우에

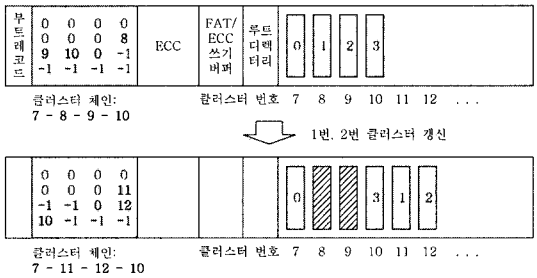

그림 5 클러스터 교체(cluster swapping)

는 오류 발생 후 안전한 복구를 위해 작업이 완료될 때까지 기존 데이타를 유지해야 하기 때문에 교대 사상을 통해 데이타 쓰기가 처리되어야 한다. 이때는 블록 단위 사상의 비효율성과 교대 영역에서의 블록 할당 비용, 그리고 주소 사상 정보를 갱신하는 비용 등으로 인해 성능 저하를 감수해야 한다. 따라서 TFFS의 파일 시스템 모듈은 파일의 내용이 갱신되는 작업에서도 정적 사상을 통해 직접적인 기록이 가능하도록 클러스터 교체(cluster swapping) 기법을 제공한다. 그림 5는 클러스터 교체 기법이 동작하는 예를 보여준다. 루트 디렉터리에 저장된 어떤 파일이 7번부터 10번까지 네 개의 클러스터로 구성되어 있다. 그림의 FAT에는 0번부터 15번까지 16개의 FAT 엔트리가 표시되어 있는데, 7번 엔트리부터 차례로 자신의 다음 클러스터 번호인 8, 9, 10이 적혀있고 10번 엔트리에는 파일의 마지막 클러스터임을 표시하기 위해 0이 적혀있다. 이에 따라 해당 파일은 7→8→9→10으로 구성된 클러스터 체인을 이루게 된다. 이때 파일의 두 번째와 세 번째 클러스터에 저장된 데이타가 변경된다면 파일 시스템은 기존의 8번과 9번 클러스터를 갱신하지 않고 11번과 12번 클러스터를 새로 할당 받아 변경된 데이타를 기록한다. 그리고 이에 따라 변경된 클러스터 체인(7→11→12→10)을 반영하여 FAT 정보를 갱신한다. 결국 FAT 섹터를 갱신하는 비용이 추가되지만 대부분의 파일 쓰기 작업을 정적 사상에 따라 고속으로 처리할 수 있다. 따라서 파일이 갱신되는 경우에도 클러스터 교체 기법을 통해 파일이 확장되는 경우와 같이 쓰기 성능을 개선할 수 있다. 단, 파일의 첫 번째 클러스터가 갱신되는 경우에 클러스터를 교체하면 파일이 포함된 디렉터리의 내용까지 연쇄적으로 갱신해야 하기 때문에 클러스터 교체 기법을 적용하지 않는다.

FAT 기반의 파일 시스템에서는 FAT 정보의 갱신이 빈번하게 발생한다. TFFS에서는 이러한 갱신 작업도 교대 사상을 통해 안전하게 처리해야 한다. 이때 갱신되는 데이타의 크기가 비교적 작고 그 위치도 연속적이지 않기 때문에 교대 사상으로 처리하는 비용이 많이 든다.

이것은 FAT 정보의 일부가 변경될 때마다 교대 사상을 통해 기록하면 그때마다 변경된 정보가 포함된 FAT 클러스터 전체를 원자적으로 갱신하기 때문이다. TFFS에서는 별도의 파일 시스템 영역으로서 FAT 쓰기 버퍼(write buffer)를 할당하여 FAT 정보의 빈번한 갱신으로 인한 성능 저하를 보완한다. FAT 쓰기 버퍼 영역의 클러스터에는 이전 데이타의 갱신 없이 섹터 단위의 순차적인 쓰기만 가능하다. 따라서 FAT 쓰기 버퍼에 대한 데이타 쓰기는 정적 사상에 따라 비원자적으로 처리할 수 있다. 이에 따라 FAT 정보가 갱신될 때 변경되는 FAT 섹터만을 FAT 쓰기 버퍼에 기록하면 기존의 FAT 정보가 안전한 상태로 유지되기 때문에 파일 시스템의 신뢰성에는 문제가 없다. 그리고 이러한 쓰기는 교대 사상을 이용하지 않기 때문에 고속으로 처리할 수 있다. 한편, FAT 정보를 참조할 때는 FAT 쓰기 버퍼를 먼저 검색하여 최신의 FAT 섹터를 찾고, 원하는 데이타가 존재하지 않는 경우에는 기존의 FAT 영역에서 해당하는 FAT 섹터를 읽는다. FAT 쓰기 버퍼가 모두 채워지면 버퍼 내에서 유효한 섹터들을 검색하여 한꺼번에 기존 FAT 정보를 갱신하고 버퍼를 비운다. 이러한 버퍼 비움 작업은 파일 시스템이 새로 기동될 때도 수행함으로써 비어있는 상태의 FAT 쓰기 버퍼를 가지고 이후 작업을 처리할 수 있도록 한다. TFFS에서 모든 데이타 섹터마다 별도로 유지되는 ECC 정보도 FAT 정보와 유사한 쓰기 특성을 가지기 때문에 동일한 쓰기 버퍼를 통해 ECC 갱신 작업을 처리할 수 있다.

4. 성능 측정 및 비교 평가

본 논문에서 설계한 플래시 메모리 파일 시스템(TFFS)의 성능을 측정하기 위하여 NAND 형 플래시 메모리를 사용하는 내장형 시스템 개발 보드에 실제로 구현하였다. 구현에 사용한 아이지시스템의 MBA-2410 개발 보드는 ARM920T CPU 코어와 NAND 형 플래시 메모리 컨트롤러를 내장한 삼성 S3C2410 MCU를 장착하고 있으며 스마트미디어 카드(SMC) 소켓을 통해 NAND 형 플래시 메모리를 장착할 수 있다[8]. 또한 기존 플래시 메모리 파일 시스템과의 비교 측정을 위해 JFFS2와 YAFFS를 동일한 환경에 이식하였다. 두 파일 시스템은 리눅스 운영체제에서 동작하기 때문에 TFFS도 리눅스의 가상 파일시스템(VFS) 구조에 통합되도록 구현하였다.

소형 이동 기기에서는 시스템 자원이 비교적 적기 때문에 플래시 메모리 파일 시스템의 크기와 메모리 사용량을 최소화해야 한다. 또한 전원 입력 후 사용자의 즉각적인 사용이 요구되는 특성으로 인해 플래시 메모리 파일 시스템의 초기 구동 시간도 최소화해야 한다.

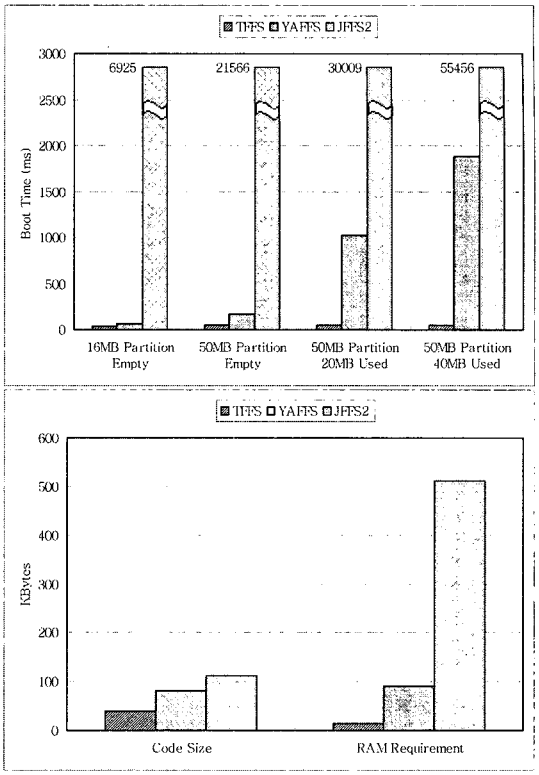

그림 6 파일 시스템 초기 구동 시간 (boot time) 및 메모리 사용량

TFFS는 기존 구현에 비해 소형 이동 기기에서 요구되는 특성을 보다 충실하게 구현하고 있다. 그림 6은 파일 시스템 초기 구동 시간 및 메모리 사용량을 비교하여 보여준다. TFFS는 단순한 주소 사상 구조에 따라 오류 복구비용이 매우 작기 때문에 초기 구동 시간은 파일 시스템의 크기나 저장된 데이타의 상태에 관계없이 매우 짧다(수십 milliseconds). 반면에 초기 구동 시 전체 파일 시스템 영역을 검색해야 하는 YAFFS와 JFFS2는 파일 시스템의 크기가 커질수록 그리고 저장된 데이타의 크기가 클수록 초기 구동 시간이 급격하게 커진다.

메모리 사용량 측면에서도 TFFS는 소형 이동 기기에 적합한 특성을 가지고 있다. 그림 6의 메모리 사용량은 파일 시스템의 크기가 16MB이고 저장된 파일의 개수가 100개인 것을 가정한 경우의 결과로서 TFFS는 YAFFS나 JFFS2 보다 코드의 크기나 RAM 사용량이 매우 적다. 특히, TFFS는 파일 시스템의 크기나 상태에 관계없이 항상 일정한 크기의 메모리만을 사용하는데 YAFFS나 JFFS2는 파일 시스템과 저장된 파일의 개수가 많을수록 메모리 사용량이 급격하게 증가한다.

플래시 메모리는 고유의 동작 특성으로 인해 데이타의 읽기보다 쓰기에 많은 시간을 필요로 한다. 따라서

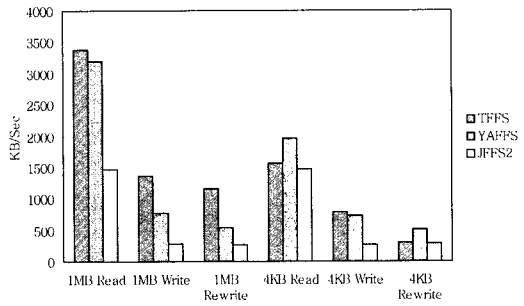

그림 7 파일 읽기 및 쓰기 성능

플래시 메모리 파일 시스템의 성능은 주로 파일의 쓰기 성능에 의해 좌우된다. 그리고 고유의 주소 재사상 기법에 따라 자유 블록을 확보하기 위한 추가 비용 때문에 플래시 메모리에 저장된 유효한 데이타의 비율(파일 시스템의 활용률(utilization))이나 데이타의 저장 상태에 따른 가용 공간의 분포에 의해 파일 시스템의 쓰기 성능은 영향을 받는다. 그림 7은 파일 시스템이 초기화된 직후의 파일 읽기 및 쓰기 성능을 측정한 결과이다. 일반적으로 파일 시스템이 초기화된 상태에서는 새로운 자유 블록을 확보하기 위한 추가 비용 없이 파일 쓰기 작업을 수행할 수 있기 때문에 그림 7은 세 가지 플래시 메모리 파일 시스템의 최대 성능을 나타낸다고 할 수 있다.

파일 시스템의 초기화 직후에 파일을 생성하여 데이타를 쓰는 시간, 다시 그 파일의 데이타를 갱신하여 쓰는 시간, 그리고 쓰여진 파일의 데이타를 읽는 시간을 측정하였다. 또한 크기가 1MB와 4KB인 파일에 대해 동일한 작업을 수행함으로써 한번에 연속적으로 접근되는 데이타의 크기에 따른 성능 변화를 측정하였다. 성능 수치는 파일 시스템의 초기화와 파일의 생성, 갱신, 읽기 작업을 20회 반복 수행하고 측정한 후 평균을 계산한 값이다. 큰 파일의 쓰기 성능에서 TFFS는 1350KB/sec의 성능을 보이며, YAFFS는 770KB/sec, JFFS2는 270KB/sec의 성능을 보였다. 실제 구현에서 플래시 메모리 파일 시스템의 성능은 하드웨어 자체의 플래시 메모리 접근 성능에 크게 의존적이다. MBA-2410 개발 보드에서 파일 시스템을 배제하고 측정한 플래시 메모리의 소거 및 기록 성능이 1550KB/sec임을 고려하면 이를 기반으로 하는 TFFS의 파일 쓰기 성능은 매우 우수하다고 할 수 있다. 이와 같은 고속의 쓰기 성능은 대용량의 멀티미디어 데이타 처리가 빈번한 최근의 소형 이동 기기에서 필수적이다. 기존 파일을 갱신하는 경우에도 TFFS는 클러스터 교체 기법에 의해 거의 동일한 쓰기 성능을 보인다. YAFFS와 JFFS2도 파일 시스템이 초기화된 직후에는 남아 있는 자유 블록을 활용하기 때문에 파일의 갱신 성능이 비교적 동일하게 유지된다. 크기가 작은 파일을 쓰는 경우에는 한번에 쓰는 데이타의 양에 비해 파일 시스템에서의 처리 비용이 크기 때문에 전반적으로 성능이 감소한다. TFFS는 주소 사상의 단위가 블록(16KB)이고 이보다 작은 파일에 대해서 클러스터 교체 기법을 적용할 수 없기 때문에 작은 파일을 갱신하는 성능이 비교적 크게 감소한다.

파일 시스템의 성능을 객관적으로 평가하기 위해서는 알려진 벤치마크를 사용하는 것이 일반적이다. 그러나 디스크 저장 장치를 기반으로 개발된 기존의 벤치마크는 소형 이동 기기에서 필요로 하는 파일 시스템의 특성을 정확히 반영하지 못한다. 벤치마크를 구성하는 작업의 내용이 소형 이동 기기에서 수행되는 것과 상이하며, 특히 헤드의 이동이나 디스크의 회전 대신에 데이타 기록을 위해 자유 블록을 확보해야 하는 플래시 메모리의 동작 특성을 고려한 테스트 항목을 포함하고 있지 않기 때문이다. 따라서 본 논문에서는 그림 8을 통해 비교적 소형 이동 기기에 적합한 작업 내용을 가지는 Bonnie 벤치마크의 수행 결과를 보여줌으로써 객관성 있는 성능 평가 자료를 제시한다. Bonnie 벤치마크는 주어진 크기의 파일에 대해 6가지 작업을 수행하고 그것의 결과를 데이타 입출력 성능으로 표시하는 대표적인 파일 시스템 벤치마크이다. 그림 8에서는 10MB 크기의 파일에 대해 탐색(seek) 시간을 제외하고 다음과 같은 5가지 작업의 결과 성능을 표시한다.

1. putc: 파일을 생성하여 1 바이트씩 순차적으로 쓰기
2. rewrite: 위 파일의 처음부터 천크(chunk, 16KB) 단위로 읽고 다시 쓰기
3. fastwrite: 위 파일 지우고 새로운 파일을 생성하여 천크 단위로 순차적으로 쓰기
4. getc: 위 파일의 처음부터 1 바이트씩 순차적으로 읽기
5. fastread: 위 파일의 처음부터 천크 단위로 순차적으로 읽기

그림에서 보는 것과 같이 Bonnie 벤치마크의 putc 및 fastwrite 성능은 각각 그림 7의 4KB 및 1MB 파일 쓰기 성능과 유사한 결과를 보여준다. 단, JFFS2는 자체적인 데이타 압축 기법을 포함하고 있기 때문에 단순히 연속된 숫자로 구성된 데이타를 사용하는 Bonnie 벤치마크에서는 압축 효과로 인해 상대적으로 우수한 성능을 보여준다. 읽기를 포함하는 rewrite와 getc에서는 TFFS의 성능이 상대적으로 낮게 나타난다. 이것은 파일 시스템 메타 데이타 입출력이 많이 수행되는 작업에서 TFFS가 작은 크기의 데이타를 처리하는 데 비효율

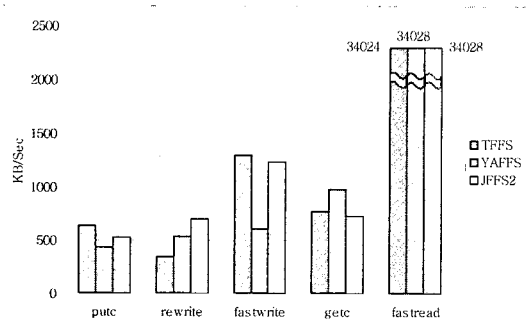

그림 8 Bonnie 벤치마크 수행 성능

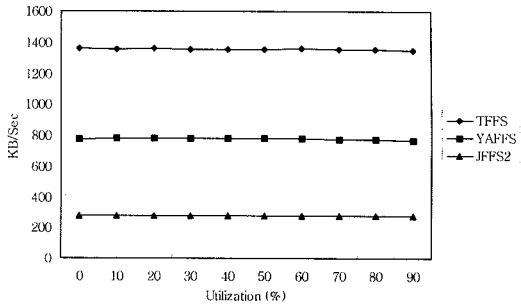

그림 9 파일 시스템의 활용률(utilization)에 따른 쓰기 성능

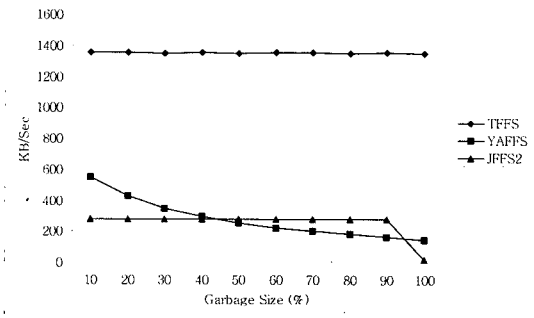

그림 10 쓰레기 수집(garbage collection)의 영향에 따른 쓰기 성능

적이기 때문이다. 용량이 큰 멀티미디어 데이타를 주로 처리하는 최근의 소형 이동 기기에서 이러한 단점이 미치는 영향이 크지는 않지만 TFFS 구현에서 추가의 최적화가 필요한 부분이다. 마지막으로 fastread 성능은 리눅스 시스템에서 기본적으로 제공하는 버퍼 캐시(buffer cache)의 효과로 인해 실제 파일 시스템의 성능과는 상관없이 매우 높은 성능 수치를 보인다.

플래시 메모리를 기반으로 하는 저장 장치에서는 파일 시스템의 활용률이 커질수록 자유 블록을 확보하기 위한 비용이 증가하여 쓰기 성능이 감소하는 것이 일반적이다. 이러한 특성은 일관성 있는 성능을 기대하는 응용 프로그램에게 바람직하지 않기 때문에 TFFS는 정적인 주소 사상 구조를 통해 파일 시스템의 상태에 상관없이 일정한 쓰기 성능을 제공한다. 그림 9는 파일 시스템의 활용률(utilization)에 따른 큰 파일의 쓰기 성능 변화를 보여준다. 실험하고자 하는 각각의 파일 시스템 활용률만큼 파일을 생성하여 채운 후에 1MB 크기의 파일을 생성하여 쓰는 시간을 측정하였다. 그림에서 TFFS 뿐만 아니라 YAFFS와 JFFS도 사용률에 관계없이 일정한 쓰기 성능을 보여주고 있다. YAFFS와 JFFS2는 고유의 쓰레기 수집(garbage collection) 기법에 따라 자유 블록을 확보하는데 그림 9의 실험에서는 파일 시스템에서 소거된 영역(garbage)이 없는 상태로 활용률만 변경하면서 파일 쓰기 작업을 수행했기 때문으로 분석된다.

일반적으로 플래시 메모리 파일 시스템에서 자유 블록을 확보하는 비용은 파일 시스템의 활용률뿐만 아니라 데이타의 저장 상태에 따른 가용 공간의 분포에도 영향을 받는다. 즉, 플래시 메모리 상에서 유효한 데이타가 분산된 정도가 클수록 이후의 쓰기 작업을 위해 새로운 자유 블록을 확보하는 비용이 커진다. 플래시 메모리 파일 시스템에서는 구현 기법에 따라 이러한 영향을 최소화하여 일관성 있는 성능 특성을 제공할 수 있어야 한다. TFFS는 클러스터의 크기를 물리 블록의 크기와 동일하게 설정하고 주소 사상을 블록 단위로 수행함으로써 파일 시스템의 사용 시간이 길어지더라도 블록 내부에서의 데이타 단편화가 발생하는 것을 근본적으로 방지한다. 따라서 플래시 메모리 상에 분산된 데이타의 저장 상태에 관계없이 초기화 때와 동일한 쓰기 성능을 제공한다. 반면에 YAFFS나 JFFS2는 이러한 경우 고유의 쓰레기 수집(garbage collection) 동작을 수행하여 자유 블록을 확보해야 하기 때문에 파일 시스템의 상태에 따라 쓰기 성능이 저하된다. 그림 10은 기존 파일을 삭제하여 소거된 영역(garbage)이 존재할 때 1MB 크기의 파일을 생성하여 기록하는 쓰기 성능을 측정한 결과를 보여준다. 실험하고자 하는 각각의 크기만큼 파일을 생성하고 모두 삭제함으로써 파일 시스템의 활용률은 0이지만 내부적으로 쓰레기 영역이 존재하도록 설정한 다음에 파일의 쓰기 성능을 측정하였다. TFFS는 소거된 영역의 존재에 관계없이 초기화 직후와 동일한 쓰기 성능을 제공한다. 그러나 YAFFS는 쓰레기 수집 작업량(삭제된 파일의 크기)에 따라 쓰기 성능이 150KB/sec까지 감소함을 볼 수 있다. JFFS2도 파일 시스템의 전체 영역이 쓰여 지고 모두 소거된 다음에는 급격하게 성능이 떨어진다. YAFFS와 JFFS2에서

의 이러한 쓰기 성능 변화는 쓰레기 수집 기법의 구현 방식에 따르는 것으로서 파일 시스템의 상태에 따라 불규칙적으로 나타난다. 결국, 파일 시스템이 일정 시간 사용된 후에는 초기화 직후와 같이 일관적인 성능을 보여주지 못하는 것을 알 수 있다. 계산 능력이 비교적 낮은 소형 이동 기기에서는 파일 시스템의 성능 변화가 응용 프로그램의 동작에 직접적으로 영향을 줄 수 있기 때문에 TFFS와 같이 항상 일정한 쓰기 성능을 제공하는 것이 중요하다.

5. 결 론

본 논문에서는 소형 이동 기기와 NAND 형 플래시 메모리에 최적화된 플래시 메모리 파일 시스템을 설계하고 구현하였다. 정적 사상과 교대 사상으로 이원화된 주소 사상 기법을 바탕으로 파일 시스템 수준에서 플래시 메모리의 동작 특성을 고려하여 데이타 쓰기를 처리함으로써 고속의 파일 쓰기 성능을 제공한다. 또한 파일 시스템의 활용률이나 데이타의 저장 상태에 관계없이 항상 일정한 쓰기 성능을 제공한다. 소형 이동 기기를 위한 구현 목표에 따라 메모리 등의 시스템 자원 사용량을 최소화하였고, 안전하고 신속한 오류 복구 기능을 통해 빠른 초기 구동 기능을 제공한다.

본 논문에서 구현된 플래시 메모리 파일 시스템은 플래시 메모리의 블록 크기와 파일 시스템의 클러스터 크기를 동일하게 설정함으로써 정적 사상의 효율적인 운용과 클러스터 교체 기법을 비교적 단순하게 구현할 수 있었다. 그러나 이러한 클러스터 설정은 파일 시스템의 공간적 효율성과 작은 크기의 파일에 대한 쓰기 성능을 저하시키기 때문에 보다 작은 크기의 클러스터로 운용되는 파일 시스템으로 개선할 필요가 있다. 또한 최근에 양산되는 NAND 형 플래시 메모리는 네 개의 블록에 대해 동시에 기록이나 소거 작업을 수행할 수 있다. 그리고 더 나아가 최근의 플래시 메모리는 대용량화 되면서 페이지의 크기가 2K 바이트로 확장되는 추세이다. 이러한 플래시 메모리에서는 데이타 기록 성능이 월등히 증가하였기 때문에 고속의 동작 특성을 반영하여 파일 시스템의 성능을 보다 더 개선하는 연구도 필요하다.

참 고 문 헌

[1] M. K. McKusick, W. N. Joy, S. J. Leffler, R. S. Fabry, "A Fast File System for UNIX," ACM Transactions on Computer Systems, vol.2, no.3, pp. 181-197, Aug. 1984.

[2] 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, Jan. 1995.

[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*(ASPLOS-6), pp. 86-97, Oct. 1994.

[4] M-Systems Inc., "TrueFFS(True Flash File System)," http://www.msystems.com/content/Corporate/Technology/TrueFFS.asp

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

[6] 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, Feb. 1992.

[7] Aleph One Company, "YAFFS (Yet Another Flash File System)," http://www.aleph1.co.uk/yaffs/yaffs.html

[8] AIJI System Co., Ltd., "MBA-2410 User's Manual Rev.0".

배 영 현

1993년 서울대학교 컴퓨터공학과 졸업
1995년 서울대학교 컴퓨터공학과 석사
1995년~현재 서울대학교 전기컴퓨터공학부 박사과정 재학중. 관심분야는 내장형 시스템, 파일 시스템, 플래시메모리 소프트웨어 등

최 종 무

1993년 서울대학교 해양학과(이학사). 1995년 서울대학교 컴퓨터공학과(공학석사) 2001년 서울대학교 컴퓨터공학과(공학박사). 2001년~2003년 유비쿼스(주) 기술연구소 책임연구원. 2003년~현재 서울대학교 컴퓨터연구소 연구원. 2003년~현재 단국대학교 정보컴퓨터학부 컴퓨터과학전공 전임강사. 관심분야는 시스템 소프트웨어, 내장형 시스템, 파일 시스템, 유비쿼터스 컴퓨팅 등

이 동 희

1989년 서울대학교 컴퓨터공학과(공학사). 1991년 서울대학교 컴퓨터공학과(공학석사). 1998년 서울대학교 컴퓨터공학과(공학박사). 1999년~2001년 제주대학교 통신 컴퓨터 공학부 조교수. 2002년~현재 서울시립대학교 컴퓨터과학부 조교수. 관심분야는 운영체제, 플래시메모리 소프트웨어, 내장형 시스템, 고성능 저장장치 등

노 삼 혁

1986년 서울대학교 컴퓨터공학과(공학사). 1993년 메릴랜드대학교 컴퓨터과학과(박사). 1994년~현재 홍익대학교 정보컴퓨터공학부 부교수. 관심분야는 시스템 소프트웨어, 병렬처리 시스템, 실시간 시스템 등

민 상 렬

1983년 서울대학교 컴퓨터공학과 졸업 1985년 서울대학교 컴퓨터공학과 석사 1989년 University of Washington 전산학 박사. 1989년~90년 IBM T.J. Watson Research Center 객원 연구원. 1990년~92년 부산대학교 컴퓨터공학과 조교수. 1992년~현재 서울대학교 전기컴퓨터공학부 교수. 관심분야는 Computer Architecture, Parallel Processing, Computer Performance Evaluation