윈도우즈 기반 플래시 메모리의 플래시 변환 계층 알고리즘 성능 분석

(Performance Analysis of Flash Translation Layer Algorithms for Windows-based Flash Memory Storage Device)

박 원 주 † 박 성 환 † 박 상 원 ††
(Wonjoo Park) (Sunghwan Park) (Sangwon Park)

요 약 최근 디지털 카메라, MP3 플레이어, 핸드폰과 같이 이동성이 중요한 요소로 차지하는 기기들이 많이 등장하였다. 이에 따라 소형화, 대용량화, 저 전력화, 비휘발성, 고속화 그리고 충격에 강한 저장장치가 필요하게 되었다. 플래시 메모리는 이러한 요구사항을 만족시키는 저장장치이다. 플래시 메모리는 하드웨어적 특성으로 인해 쓰기 전 소거(erase-before-write)연산을 수행해야만 한다. 따라서 플래시 메모리를 효과적으로 동작시키기 위해서 FTL이 필요하다. FTL은 플래시 메모리의 단점을 보완해주면서 상위 파일 시스템을 그대로 사용할 수 있는 장점을 가진다. 따라서 차후 디스크는 플래시 메모리로 대체될 것이다. 대부분의 PC에서 윈도우즈 기반의 OS를 사용하기 때문에 기존 FTL이 윈도우즈 기반의 OS에서 어떠한 성능을 보이는지 분석할 필요가 있다. 본 논문에서는 실험속도를 빠르게 하기위해 FTL 성능분석도구를 개발한다. 이를 이용하여 여러 FTL 알고리즘들이 윈도우즈 기반의 OS의 디스크 I/O를 처리하는 성능을 분석한다. FTL의 성능은 매핑 방법, 한 블록 내에 섹터를 기록하는 방법과 덮어쓰기의 처리방법 등을 분석하여 비교가 가능하다. 실험한 FTL중 개선된 로그 블록 기법이 실험 결과 중에 가장 좋은 성능을 보인다. 따라서 차후 디스크가 플래시 메모리로 대체 될 경우, 로그 블록 기법을 잘 적용 시켜야 할 것이다.

키워드 : 플래시 메모리, 플래시 변환 계층

Abstract Flash memory is widely used as a storage device for potable equipments such as digital cameras, MP3 players and cellular phones because of its characteristics such as its large volume and nonvolatile feature, low power consumption, and good performance. However, a block in flash memories should be erased to write because of its hardware characteristic which is called as erase-before-write architecture. The erase operation is much slower than read or write operations. FTL is used to overcome this problem. We compared the performance of the existing FTL algorithms on Windows-based OS. We have developed a tool called FTL APAT in order to gather I/O patterns of the disk and analyze the performance of the FTL algorithms. It is the log buffer scheme with full associative sector translation(FAST) that the performance is best.

Key words : Flash Memory, Flash Translation Layer

1. 서 론

최근 디지털 카메라, MP3 플레이어, 핸드폰과 같이 이동성이 중요한 요소로 차지하는 기기들이 많이 등장하였다. 이에 따라 소형화, 대용량화, 저 전력화, 비휘발성, 고속화 그리고 충격에 강한 저장장치가 필요하게 되었다. 플래시 메모리는 이러한 요구사항을 만족시키는 저장장치이기 때문에 디스크에 비해 많은 장점을 가지고 있다. 따라서 플래시 메모리는 점차 디스크를 대체하여 대표적인 모바일 기기의 대용량 저장장치로도 자리매김을 할 것이다.

플래시 메모리는 임의의 위치의 섹터를 읽고 쓸 수 있는 디스크와 다르게 쓰기 전 소거(erase-before-write) 연산을 한다. 이것은 플래시 메모리의 하드웨어적 특성으로 섹터에 쓰기 연산을 하기 전에 그 섹터가 속한 일

· 본 논문은 2007년 한국외국어대학교 교내 학술연구지원비의 지원을 받아 연구되었음
† 학생회원 : 한국외국어대학교 컴퓨터및정보통신공학과
wjpark@dislab.hufs.ac.kr
shpark@dislab.hufs.ac.kr
†† 종신회원 : 한국외국어대학교 컴퓨터및정보통신공학과 교수
swpark@dislab.hufs.ac.kr
논문접수 : 2007년 4월 27일
심사완료 : 2007년 8월 8일

정 크기의 여러 섹터를 소거 연산을 통해 초기화한 다음 쓰기 연산을 수행해야 한다는 것이다. 또한 플래시 메모리는 섹터에 대한 읽기와 쓰기, 소거의 처리속도가 다르다. 특히 읽기와 쓰기에 비해 소거 연산은 매우 느려 자주 소거를 하게 되면 플래시 메모리의 성능이 저하된다.

따라서 플래시 메모리는 성능 향상을 위해 FTL을 사용해야한다. FTL은 어떠한 섹터에 대해 쓰기를 요구하면 FTL 내부의 매핑테이블을 이용하여 요구한 섹터와 실제 기록하는 섹터를 다른 위치에 쓰도록 한다. 이는 플래시 메모리의 성능 저하의 원인인 소거 연산을 가능한 적게 발생하지 않도록 해주어 성능 저하를 막는다. 또한 FTL은 플래시 메모리를 운영체제에서 보았을 때 일반 디스크와 같은 상태로 볼 수 있게 한다. 따라서 플래시 메모리가 디스크를 대체하기 위해서는 FTL이 필수적이다.

본 논문에서는 플래시 메모리를 블록 저장 장치로 보여주는 여러 가지 FTL의 성능을 평가한다. 이를 위해서 범용으로 사용되는 운영체제인 윈도우즈의 파일 시스템에서 디스크 I/O 패턴을 추출하기 위해 윈도우즈 필터 드라이버를 작성하여 파일 시스템에서 디스크로 보내지는 I/O 패턴을 얻어 실험에 사용했다. 또한 실험을 위한 시뮬레이션 도구로 실제 플래시 메모리처럼 동작하는 가상 플래시 메모리와 각 FTL 알고리즘을 구현하였다. 시뮬레이션 도구에서 얻어지는 정보들을 통해 어떤 FTL 알고리즘의 성능이 가장 좋은 알고리즘인지 평가할 수 있었다.

본 논문에서는 소거 연산의 횟수와 읽기와 쓰기, 소거 연산의 수행 시간을 이용하여 결과를 분석한다. 각 실험 결과에서 FTL로 한 번의 쓰기 명령이 요구되었을 때 읽기와 쓰기, 소거의 발생 횟수로 성능을 비교한다. 이 횟수를 계산하여 한 섹터의 쓰기 요구를 받았을 때 얼마나 많은 추가비용이 소요되는지를 계산한 뒤, 각 FTL 알고리즘을 비교한다.

2. 관련연구

이 장에서는 플래시 메모리의 일반적인 시스템 구조 및 플래시 메모리와 관련된 연구에 관하여 설명한다.

2.1 플래시 메모리와 FTL 시스템

그림 1은 플래시 메모리에서 FTL을 사용할 때의 시스템에 대한 구조도이다. FTL은 플래시 메모리와 파일 시스템 사이에 존재하며 파일 시스템에서 플래시 메모리를 블록 저장 장치로 바라볼 수 있게 해준다.

플래시 메모리는 크게 NAND 플래시 메모리와 NOR 플래시 메모리로 나누어진다. NOR 플래시 메모리는 주소를 통해 메모리를 접근할 수 있어 플래시 메모리에서

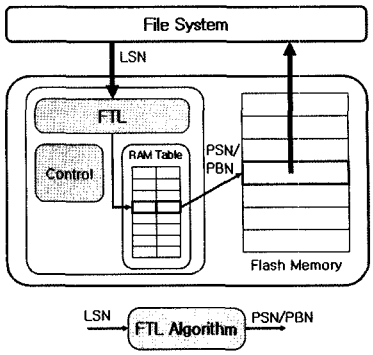

그림 1 FTL과 플래시 메모리 시스템 구조도

직접 프로그램을 수행할 수 있다. 하지만 NAND 플래시 메모리는 일정한 크기인 섹터 단위로 읽기와 쓰기 연산을 수행하고, 여러 섹터를 합한 블록 혹은 소거 단위로 소거를 수행해야한다. 본 논문에서는 NAND 플래시 메모리만을 대상으로 한다.

일반적으로 플래시 메모리에는 FTL이라는 시스템 소프트웨어가 있다. FTL은 파일 시스템으로부터 전달받은 읽기와 쓰기에 대한 연산 명령어와 섹터 번호, 길이를 요청 받으면 효율적인 매핑을 통해 전체적으로 플래시 메모리의 읽기와 쓰기, 소거 연산을 줄여서 해당 연산을 빠르게 수행한다. FTL 알고리즘은 플래시 메모리의 내부 롬(ROM)에 기억되고 FTL의 매핑 테이블은 플래시 메모리의 램(RAM)에 저장하며 이를 통하여 파일시스템에서 요청하는 논리적인 섹터를 플래시 메모리의 물리적인 섹터로 매핑 할 수 있게 한다.

2.2 FTL

FTL[1]은 읽기와 쓰기, 소거의 속도차이, 쓰기 전 소거와 소거횟수 제한 등의 플래시 메모리의 특징을 잘 고려하여 플래시 메모리의 성능 향상과 수명 연장을 주 목적으로 한다. FTL은 그림 1에서처럼 롬에 저장되어 플래시 메모리가 동작할 때 주로 램 테이블과 실제 플래시 메모리에 저장된 여러 가지 정보를 이용하여 매핑 테이블을 유지 및 관리하는 역할을 수행한다[2].

FTL 알고리즘에서 사용되는 매핑은 섹터매핑, 블록매핑과 하이브리드(hybrid)매핑으로 나누어진다. 섹터매핑은 섹터 단위로 매핑 테이블을 만들어 물리적인 섹터 번호와 논리적인 섹터 번호를 매핑하고, 블록매핑은 물리적인 블록 번호와 논리적인 블록 번호를 매핑한다. 섹터매핑을 하면 한 번의 매핑 테이블의 검색으로 실제 플래시 메모리의 물리적인 섹터 번호를 알 수 있어서 매우 빠르게 동작할 수 있지만 플래시 메모리의 용량이 증가하는 만큼 매핑정보를 저장하기 위하여 많은 양의 램을 사용한다. 블록매핑은 플래시 메모리의 블록의 수만큼의 테이블을 램에 저장하여 섹터매핑보다 상대적으

로 적은 양의 램을 사용하지만 덮어쓰기 연산을 효율적으로 수행할 수 없다. 하이브리드 매핑은 블록매핑을 하여 램의 사용량을 줄이는 한편, 블록 내에서는 섹터매핑을 하여 두 가지 매핑의 장점을 모두 취하는 방법이다.

플래시 메모리에서는 한 개의 블록 내에 있는 섹터들에 대해 기록하는 방법을 두 가지로 나눈다. 한 블록에 저장되는 섹터들이 정해진 위치(offset)에 저장되는 방식은 고정 섹터방식(in-place)이고, 섹터들이 블록 내의 임의의 위치에 저장되는 방식을 변동 섹터 방식(out-of-place)이다.

또한 각 FTL 알고리즘은 내부적으로 기본적인 읽기와 쓰기, 소거 연산 외에 필요에 따라 할당(allocate) 연산이나 교환(switch) 연산, 합병(merge) 연산 등을 사용한다. 할당 연산은 빈 블록이 필요할 경우 빈 블록을 탐색하여 반환하는 연산이다. 교환 연산은 새로운 블록을 할당받아 기록한 내용이 기존의 논리적인 블록에 해당하는 블록 내용보다 모두 최신의 정보일 경우 기존에 기록된 블록을 소거 하고, 블록매핑 테이블의 값을 새로운 블록으로 바꾸는 연산이다. 합병 연산은 두 개의 블록에서 최신 정보만 복사하여 새로운 블록에 쓰고, 매핑 테이블을 새로운 블록으로 바꾸어 준 다음 합병에 사용된 두 개의 블록을 소거하는 연산이다.

효율적인 FTL 알고리즘은 많이 만들어져 있다[3]. 삼성에서 개발한 알고리즘[4]은 하이브리드 매핑방법을 사용하는 알고리즘으로 각 섹터의 부가 영역(spare space)을 모아 읽기를 많이 수행하더라도 소거와 쓰기를 줄이도록 작성된 알고리즘이다. Lexar Media에서 만들어진 알고리즘[5]은 블록매핑을 하고 덮어쓰기에 대해 고정 섹터 방식으로 섹터를 기록하여 블록과 블록을 연결 리스트로 연결하는 방법으로 소거를 줄이는 알고리즘이다. 본 논문에서는 위의 여러 가지 매핑방법과 기록방법, 추가연산 등을 이용하는 여러 FTL 알고리즘 중 특징이 분명한 네 가지에 대하여 그 특성을 분석하였다. 선정된 알고리즘은 Mitsubishi사의 알고리즘(MITS)[6]과 M-System의 FMAX[7], 로그블록을 사용한 알고리즘(로그블록 기법[8]과 FAST[9])이다.

이외에도 플래시 메모리용 파일시스템이 있다. 이러한 파일시스템으로 로그-구조형 파일시스템(log-structured file system)[10]이 있다. 플래시 메모리는 쓰기 전 소거의 특성을 가지므로 로그를 추가하여 덮어쓰기를 방지하는 방식이 플래시 메모리에 적합하다. JFFS(journaling flash file system)[11]는 임베디드 리눅스에서 사용할 수 있도록 만들어진 파일 시스템이다. 후에 JFFS2[12]로 발전된 이 파일 시스템은 파일을 로그를 기반으로 하는 노드의 리스트로 이루어진다. 대부분의 노드는 파일의 메타데이타의 복사본과 파일내의 데이타의 길이

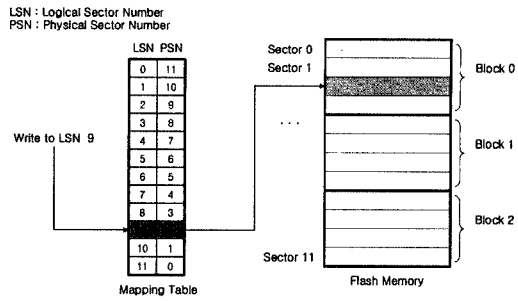

그림 2 섹터매핑 구조도

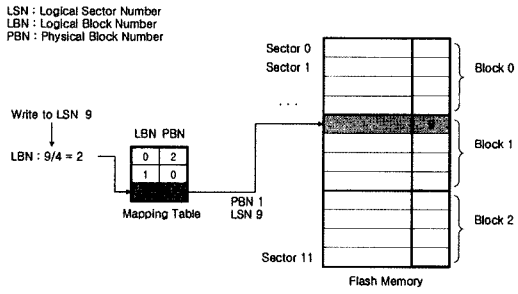

그림 3 블록매핑 구조도

를 가진다. 또, 특별한 디렉토리-엔트리 노드를 파일 이름과 관계된 메타데이타로 구성하여 파일시스템이 디렉토리-트리를 가지도록 하고, 메타데이타에 버전 정보를 기록하여 최신의 메타데이타를 찾아 시스템을 유지한다. YAFFS(yet another flash filing system)[13]는 파일에 대해 일정한 크기의 덩어리(chunk)를 구성하여 헤더와 함께 저장한다. 헤더는 덩어리의 정보를 저장하고 파일은 하나의 헤더 덩어리와 데이타 덩어리로 이루어진다. 이러한 파일을 연결하는 디렉토리 덩어리를 만들어서 파일시스템을 유지한다. 이외에도 많은 플래시 전용 파일시스템이 존재하고, 플래시 메모리를 이용한 메인 메모리나 플래시 메모리 위의 응용프로그램 데이타 구조 등 다양한 분야에서 활용되고 있다.

본 논문에서는 파일 시스템의 변경없이 플래시 메모리를 사용할 수 있는 FTL에 대해서 비교 분석 할 것이다.

3. FTL 알고리즘

이 장에서는 본 논문에서 실험하는 FTL에 관하여 설명한다.

3.1 Mitsubishi사의 알고리즘(MITS)

Mitsubishi의 MITS 알고리즘은 그림 4의 램테이블(Ram-table)에 블록매핑을 하고 플래시 메모리의 한 블록을 데이타 공간(data space)과 여유 공간(spare space)

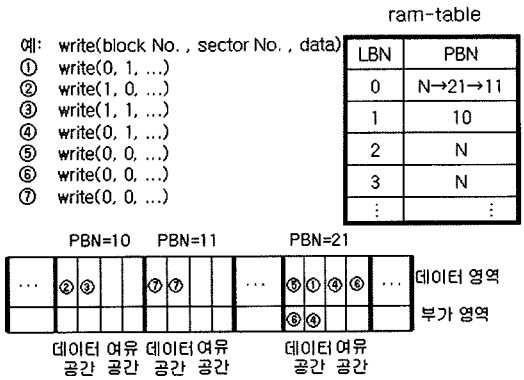

그림 4 Mitsubishi사 알고리즘 구조도

으로 나누어 덮어쓰기에 대한 해결책을 제시한다. FTL은 블록내의 여유 공간을 제외한 데이타 공간을 논리적인 한 블록으로 보고 요청에 주어진 논리적 섹터 번호(logical sector number)를 논리적인 블록의 크기로 나누어 논리적 블록 번호(logical block number 이하 LBN)를 계산하고 이 값을 이용하여 램 테이블로부터 물리적 블록 번호(physical block number 이하 PBN)를 찾아낸다. 그림 4는 한 블록이 소거 단위이고, 한 블록에 4개의 섹터가 기록될 때를 가정하여 알고리즘이 동작하는 예제를 그린 그림이다. 이하 알고리즘에 대한 예제도 같은 가정을 한다.

FTL로 쓰기 요청에 LBN에 해당하는 PBN가 없는 경우(①) 빈 블록을 찾아 LBN와 PBN을 램테이블에 매핑한다(LBN 0이 N에서 21로 변경). 데이타는 고정 섹터 방식으로 데이타 공간의 데이타 영역에 기록하고, 논리적 번호와 블록 상태 등은 해당 부가 영역(spare area)에 기록한다(PBN 21의 1번 섹터 기록). 논리적 블록 번호에 해당하는 물리적 블록 번호가 있다면(②) 기록될 섹터의 위치를 읽는다. 읽은 섹터가 비어 있는 경우에는 데이타를 기록한다(PBN 10의 0번 섹터 기록). 섹터가 비어 있지 않아서 덮어 쓰기가 발생 할 경우(④) 해당 여유 공간에 변동 섹터 방식으로 기록하고 데이타 공간의 이전 내용을 사용 불가(invalid) 상태로 기록한다(PBN 21의 1번 섹터 사용불가 기록, 2번 섹터 기록). 만약 블록의 여유 공간이 모두 사용되어 빈 여유 공간이 없을 경우(⑦) 합병 연산을 수행한다. 합병연산은 빈 블록을 할당한 후, 기존 블록의 여유 공간의 데이타 중 가장 최신 데이타들을 찾아 고정 섹터 방식으로 기록하고, 기존 블록의 데이타 공간의 유효한(valid) 데이타를 찾아 새로 할당받은 블록에 기록한다. 그리고 새로 할당받은 블록에 대한 정보를 램 테이블에 갱신하고 기존 블록을 소거하는 순으로 동작한다(LBN 0이 21에서 11로 변경).

이 알고리즘은 여유 공간만큼 실제 플래시 메모리의 블록 수를 늘려야 하기 때문에 논리적인 플래시 메모리의 양과 물리적인 양이 다르다. 경제적인 입장에서 보면 여유 공간의 크기를 잘 정하는 것이 그 만큼 실제 플래시 메모리의 양을 줄일 수 있기 때문에 중요하다. 본 논문에서는 플래시 메모리의 양보다 소거 횟수나 소요시간에 대해 비교를 실시 할 것이므로 다음에 나올 알고리즘들과 같은 크기의 물리적인 플래시 메모리 양을 가지도록 여유 공간의 비율을 0.5로 하여 블록의 절반을 여유 공간으로 사용한다. 따라서 물리적인 플래시 메모리의 크기는 논리적으로 제공하는 플래시 메모리의 두 배를 가진다.

3.2 M-system(FMAX)

M-system의 FMAX 알고리즘은 블록매핑을 하고 복사 블록(mirroring block)을 사용한다. LBN을 이용하여 PBN을 찾아내는 방법은 MITS와 같다.

LBN에 해당하는 PBN이 없는 경우(①), 빈 블록을 할당하여 램 테이블에 기록하고(LBN 2이 N에서 10으로 변경) 섹터의 부가 영역에 논리 블록 번호를 기록한 후 고정 섹터 방식으로 데이타를 기록한다(PBN 10의 1

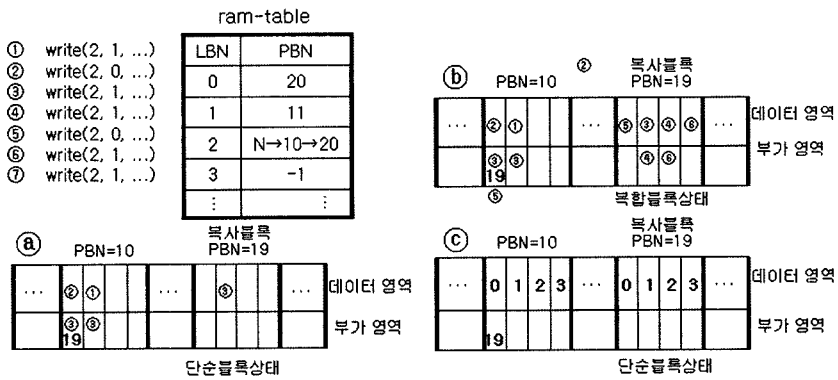

그림 5 M-system사 알고리즘 구조도

번 섹터 기록). 해당하는 블록이 존재하고 해당 섹터가 비어있으면(②), 해당 섹터에 고정섹터 방식으로 데이타를 쓴다(ⓐⓑ PBN 10의 0번 섹터 기록). 만약 해당 섹터에 데이타가 쓰여 있으면 덮어쓰기 연산이 발생한다(③). 이 경우 복사 블록이 없으면 빈 블록을 할당하여 고정 섹터 방식으로 데이타를 쓴다(PBN 10의 0번 섹터 복사블록 19 기록, 1번 섹터 사용불가 기록, PBN 19의 1번 섹터 기록). 복사블록이 고정 섹터 방식으로 데이타를 저장하고 있으면 단순 블록 상태(simple block state)이다(ⓐ). 복사블록에 기록된 섹터에 대해 다시 쓰기 요청이 들어오면(④) 복사 블록에 오프셋을 아래로 옮겨 변동 섹터 방식으로 기록한다(PBN 19의 3번 섹터 기록). 이 때 복사 블록의 상태를 복합 블록 상태(Compound block State)라고 한다(ⓑ). 만약 복사 블록에 빈 섹터가 없으면 합병 또는 교환 연산을 수행하는데 복합 블록 상태라면(ⓑ) 합병 연산을, 단순 블록 상태(ⓒ)이면 교환 연산을 수행한다. 합병은 복사블록의 유효한 데이타와 데이타 블록의 유효한 데이타를 새로운 빈 블록에 복사 하고 램테이블을 갱신하는 순서이다. 교환연산은 복사블록의 PBN을 LBN에 맞게 램테이블을 변경한다.

이 알고리즘은 모든 블록에 대해 복사블록을 가지므로 모든 블록에 대해 덮어쓰기 명령이 수행되면 논리적인 크기의 두 배에 해당하는 물리적인 플래시 메모리의 크기가 필요하다. 하지만 MITS 알고리즘과 같이 본 논문에서는 크게 관계하지 않고 같은 디스크 I/O에 대해 동작하는 성능을 비교한다.

3.3 로그 블록 기법(Log Block Scheme : BAST : Block Associative Sector Translation)

로그 블록 기법은 블록매핑을 하고 로그 블록을 이용하는 알고리즘으로 로그블록을 동적으로 할당하기 위한 로그블록테이블과 로그블록의 내용을 빠르게 접근하기 위한 로그 섹터 테이블을 램테이블에 추가한다.

쓰기 요청의 LBN에 해당하는 PBN이 없는 경우(①), 빈 블록을 할당하여 데이타를 고정 섹터 방식으로 기록한 후 해당 블록의 첫 번째 섹터에 데이타 블록이라고 기록 한다(ⓐPBN 10의 1번 섹터 기록, 0번 섹터 부가영역 기록). PBN이 있는 경우, 해당 섹터가 비어 있으면(②) 데이타를 기록하고(ⓐPBN 10의 0번 섹터 기록) 해당 섹터에 데이타가 있으면 덮어 쓰기를 수행한다. 덮어 쓰기는 로그 블록 테이블에서 해당 블록에 해당하는 로그 블록이 있는 지를 검사한다. 로그 블록이 없는 경우, 로그 블록 테이블에 할당이 안 된 로그 블록이 있는 지를 검사한다. 빈 로그 블록이 있는 경우 새로운 블록을 할당(③)하여 로그 블록으로 기록한 후 로그 블록에 변동 섹터 방식으로 데이타를 기록한다(ⓐPBN 19의 0번 섹터 기록). 로그 블록이 모두 사용되고 있다면(④), 로그 블록 중 하나를 희생 블록(victim block)으로 선정하여 합병 연산을 수행한 뒤 새로운 로그블록을 할당하여 덮어쓰기의 나머지 수행을 한다(ⓐ로그블록1을 희생 블록으로 선정, LBN 0을 로그블록 1인 PBN 15와 합병 PBN 13을 LBN 0에 할당, 로그블록 1에 LBN 2, PBN 8을 할당). 로그 블록 테이블에 덮어쓰기 할 데이타 블록의 로그 블록이 있는 경우(⑤), 그 로그 블록에 빈 섹터가 있다면 변동 섹터 방식으로 기록한다(ⓐPBN 19의 1번섹터 기록). 만약 마지막 섹터에 데이타를 기록하면(⑦) 교환 또는 합병 연산을 한다. 로그 블록에 순차적으로 데이타가 쓰여 있다(ⓑ)면 교환 연산을 수행하고, 로그 블록이 비순차적으로 되어 있으면(ⓐ) 합병 연산을 수행한다. 합병 연산은 새 블록을 할당받아 희생 로그 블록의 최신 섹터와 해당 데이타 블록의 유효한 데이타를 새로운 블록에 기록하고 기존 데이타 블록과 희생 로그 블록을 소거한다. 로그 블록 테이블에 희생 블록을 제거하고, 램 테이블을 갱신하는 순서이다.

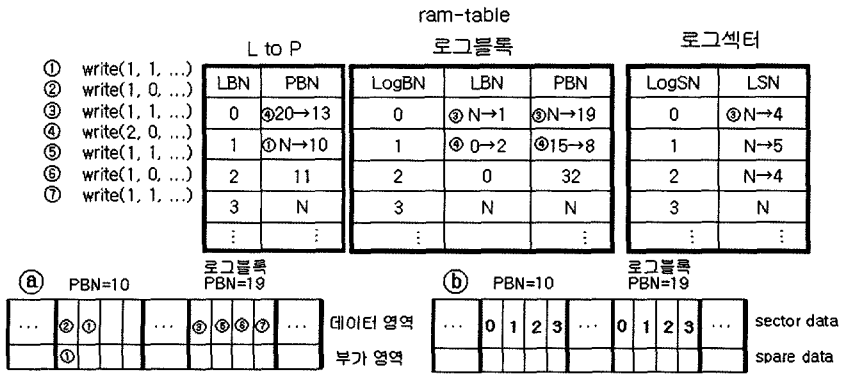

그림 6 로그블록기법 알고리즘 구조도

이 알고리즘은 물리적인 플래시 메모리의 크기가 데이타로 사용할 크기와 로그블록으로 사용할 블록 수를 더한 크기만 존재한다면 동작 가능한 알고리즘이다. 따라서 앞의 두 알고리즘에 비해 경제적이다.

3.4 개선된 로그 블록 기법(Log Block Scheme : FAST : Full Associative Sector Translation)

로그 블록 기법은 순차 쓰기용 로그 블록, 임의 쓰기용 로그 블록을 이용하는 로그블록 기법으로 램 테이블에는 순차 쓰기용 로그 블록 테이블, 임의 쓰기용 로그 블록 테이블을 추가 사용하고, 데이타 블록은 블록매핑을, 순차 쓰기용 로그 블록은 블록매핑을, 임의 쓰기용 로그 블록은 블록 매핑 및 섹터 매핑을 한다.

쓰기 연산이 발생하면, 요구 섹터에 해당하는 섹터를 먼저 임의 쓰기용 로그블록에 해당 블록의 덮어 쓰기 된 데이타가 있는지 확인한다. 임의 쓰기용 로그블록은 모든 덮어쓰기 된 섹터가 모여 있으므로 자주 덮어쓰는 섹터는 임의 쓰기용 로그블록에 데이타가 있다. 이에 대한 자세한 내용은 뒤에 설명하겠다. 임의 쓰기용 로그블록에 데이타가 없다면 순차쓰기용 로그블록 테이블을 확인하여 현재 기록하고자 하는 섹터와 같은 블록이 기록되고 있는지 확인한다. 만약 순차쓰기용 로그블록에 기록되는 블록이 현재 기록할 데이타의 블록과 같은 블록이라면 순차쓰기용 로그블록에 고정 섹터 방식으로 데이타를 기록한다(①PBN 33의 2번섹터 기록). 이 때 기록된 섹터의 오프셋이 블록의 마지막 오프셋이라면 순차쓰기용 로그 블록에 기록된 섹터의 수를 검사한다. 순차쓰기용 로그 블록의 모든 섹터에 데이타가 기록되었으면 교환 연산을 수행하고, 그렇지 않으면 합병 연산을 수행한다.

순차쓰기용 로그블록에서의 합병연산은 데이타블록에서 순차쓰기용 로그블록의 비어있는 섹터를 복사하고 순차쓰기용 로그블록을 데이타블록으로 교환한다. 이미 순차쓰기용 로그블록에 기록된 섹터에 대한 덮어쓰기가 발생한 경우는 새로운 블록에 순차쓰기용 로그블록과 데이타 블록을 합병한다. 순차쓰기용 로그블록은 FTL 내에 한 개 블록만 유지한다. 임의쓰기용 로그블록은 임의쓰기용 로그블록의 섹터들이 모두 사용될 때까지 데이타를 현재 사용한 로그블록의 처음 빈 섹터에 추가한다. 만약 임의쓰기용 로그블록에 현재 기록하고자 하는 섹터가 있다면 이전에 기록되어있던 데이타를 섹터매핑 테이블에서 제거하고 임의쓰기용 로그블록의 처음 빈 섹터에 기록한다(②PBN 19의 1번 섹터 기록). 임의쓰기용 로그블록에 현재 기록할 섹터가 없다면 데이타 블록을 확인한다. 데이타 블록에서 해당 섹터가 기록되어있다면(③) 임의쓰기용 로그블록에(PBN 19의 2번 섹터 기록), 없다면(④) 데이타 블록에 해당 섹터를 기록한다(PBN 10의 0번 섹터 기록).

임의쓰기용 로그블록의 섹터들이 모두 사용되면 원형 큐의 형태로 앞에서부터 한 블록씩 임의쓰기용 로그블록을 합병한다. 임의쓰기용 로그블록의 합병 방법은 합병 할 임의쓰기용 로그블록에 대한 섹터매핑 테이블을 참고하여 현재 합병 할 임의쓰기용 로그블록내의 섹터와 연관된 데이타블록을 찾고 각 데이타 블록을 합병한다. 이 때 임의쓰기용 로그블록의 전체에 대해 해당 데이타 블록에 속하는 섹터들을 모두 합병연산에 반영하고 기존의 데이타 블록에서 유효한 섹터를 복사 한다. 이러한 수행을 합병할 임의쓰기용 로그블록의 유용한 모든 섹터에 대해 실시하며 각 섹터에 대해 합병이 끝나면 임의쓰기용 로그블록을 소거한다.

표 1은 위의 네 가지 알고리즘의 특징을 표로 나타냈다.

4. 실 험

본 논문에서는 두 가지 실험을 한다. 실험 1은 윈도우즈기반 운영체제에서 파일을 복사 삭제할 경우 발생하는 I/O를 추출하여 우리가 작성한 FTL시뮬레이터에서 구현된 FTL을 통해 플래시 메모리에서 발생할 읽기와 쓰

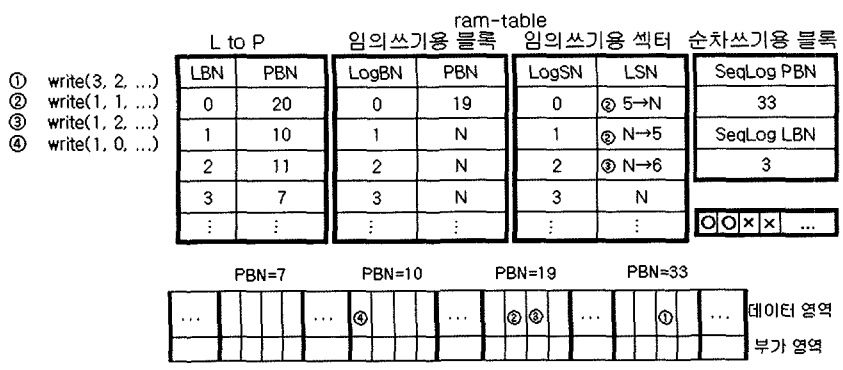

그림 7 개선된 로그블록 기법 구조도

표 1 알고리즘의 특성 비교

	MITS	FMAX	로그블록기법	FAST
데이타	in place	in place	in place	in place
매핑	block	block	Hybrid	Hybrid
덮어 쓰기	out of place	in place → out of place	out of place	in place & out of place
특징	블록 내의 spare space를 사용	각 블록 당 1:1로 복사 블록을 사용	한 블록 에 로그 블록 하나를 할당	로그블록을 순차쓰기용과 임의쓰기용으로 구분

기, 소거의 횟수를 측정한다. 실험 2는 윈도우즈 기반 운영체제에서 파일을 복사 삭제할 경우 발생하는 I/O중에 특징적인 I/O패턴과 다른 임베디드 시스템에서 얻을 수 있는 I/O패턴을 데이타로 임베디드 실험 구조를 구축한다. 임베디드 실험 구조에서 실제 NAND 플래시 메모리에 FTL을 구동시켜 FTL의 동작시간과 플래시 메모리에서 발생하는 읽기와 쓰기, 소거 횟수를 기록한다.

플래시 메모리의 읽기와 쓰기, 소거는 각 플래시 메모리마다 다른 속도를 가지고 있지만, 읽기와 쓰기, 소거의 소요시간이 차이가 나는 것은 모두 동일하다. 따라서 알맞은 비율로 소요시간을 계산할 수 있도록 플래시 메모리에서 발생하는 읽기와 쓰기, 소거의 횟수를 기록한다. 이때 FTL의 동작 비용은 제외한다.

4.1 실험 1 (시뮬레이션을 이용한 실험)

우선 첫 번째 실험에서 사용할 윈도우즈 환경에서 디스크로 가는 I/O를 추출하기 위해 I/O를 얻어오는 드라이버를 작성한다. diskiomo 드라이버(Disk I/O Monitering Driver)는 디스크 드라이버의 하위 계층 드라이버[14]이며 필터드라이버로 디스크 드라이버에서 실제 디스크로 가는 IRP를 스니핑하여 I/O를 얻는다. diskiomo 드라이버에서 얻은 I/O는 파일시스템과 디스크 드라이버를 거쳐 실제 물리 디스크에게 전송되는 정보이며 실제 플래시 메모리로 전송되는 I/O 요청과 동일하다. 따라서 diskiomo 드라이버로 얻어낸 I/O를 로그 파일(diskio.log)로 기록한다. 기록하는 방법은 읽기와 쓰기 부호(r/w) 1바이트와 I/O의 시작섹터번호(정수), 섹터의 수(정수)를 이진 데이타로 로그파일에 기록한다.

FTL APAT(FTL Algorithm Performance Analysis Tool)는 파일로 저장된 I/O 로그정보를 읽어 FTL 알고리즘의 성능을 평가하는 도구이다. FTL APAT는 크게 3가지의 객체로 나누어져 있다.[1] "diskio.log" 파일에서 로그정보를 읽어오는 LogFileReader 객체와 각각의 FTL 알고리즘을 구현해 놓은 FTL_Alg 객체, 그리고 실제 플래시 메모리처럼 동작하는 가상 플래시 메모리가 있다. FTL APAT는 LogFileReader 객체에서 읽은 로그 정보를 FTL_Alg 객체에서 수행시켜 플래시 메모리에서 발생할 읽기와 쓰기, 소거 횟수를 계산한다. FTL APAT는 FTL_Alg객체와 가상 플래시 메모리에서 얻어진 각종 통계 정보를 실시간으로 표시하며 결과인 플래시 메모리에서 발생한 읽기와 쓰기, 소거 횟수를 확인 가능한 특정 파일로 저장을 한다. 실험 시스템은 그림 8과 같다.

표 2는 diskiomo 드라이버와 FTL_APAT가 동작한 환경이다.

표 3은 드라이브 E에 I/O를 발생시키는 파일의 정보이다. 윈도우즈에서 사용되는 파일시스템에서는 다양한 워크로드를 만들어 낼 수 있지만[15], 현재 플래시 메모리는 일반적으로 디지털 카메라의 사진 정보나 MP3플레이어의 음악 파일을 저장하는 데 사용되므로 현재의 FTL 알고리즘 성능 분석을 위해서 사진 파일인 JPG와 음악 파일인 MP3를 윈도우즈 환경에서 복사 및 삭제하여 I/O를 하였다. 일반적인 PC에서 사용되는 운영체제는 멀티 프로세스 기반 시스템이므로 멀티프로세스 환경에서의 I/O를 얻어냈다.

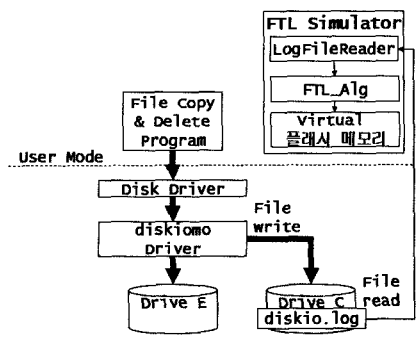

그림 8 실험 1의 실험 구조

표 2 실험 1 환경

운영체제	Windows 2000 Professional
파일시스템	FAT, NTFS
사용 디스크	드라이브 C : "diskio.log" 파일 저장 드라이브 E : diskiomo 드라이버를 부착
드라이브 E 크기	100M바이트
발생 I/O의 크기	130M바이트 정도의 자료를 이용해 약 3.5G바이트 정도의 I/O를 발생시킴

1) 자세한 내용은 http://dislab.hufs.ac.kr/flash를 참고.

표 3 I/O 발생 패턴

JPG	200K바이트~400K바이트의 작은 크기의 파일
MP3	3M~5M의 큰 크기의 파일
혼합	JPG와 MP3을 혼합하여 다중 프로세스를 통해 다량의 분할 발생

4.2 실험 2 (임베디드 보드를 이용한 실험)

두 번째 실험은 임베디드 보드에서 실제 FTL 알고리즘을 적용하여 NAND플래시에서 추출한 여러가지 I/O를 동작시키는 실험이다. 첫 번째 실험과 달리 임베디드 보드에 착탈 가능한 NAND플래시 메모리에서 동작할 수 있는 환경을 구성하고 NAND플래시 메모리의 크기만큼 디스크의 크기를 제한하여 얻어낸 로그를 이용했다. 실험 환경은 다음 표 4와 같다.

실험 2에서 사용한 I/O는 실험 1에서 사용한 NTFS 혼합형 I/O와 함께 FAST의 성능을 측정한 논문에서 사용한 디지털 카메라와 리눅스, 심비안에서 얻은 I/O이다.

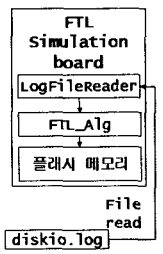

그림 9 실험 2의 실험구조

표 4 실험 2 환경

임베디드 보드	SMDK 2440 v 2.0
플래시 메모리	삼성 K9F5608U0B 32M바이트
운영체제	없음
소프트웨어	ADS v1.2 및 플래시 메모리 동작 소프트웨어(aiji system)
E 드라이브 크기	16M바이트
파일시스템	NTFS 및 디지털카메라와 리눅스, 심비안 등
발생 I/O의 크기	NTFS는 멀티 프로세스, I/O 10000회, 디지털 카메라와 리눅스, 심비안 등 일정 크기

5. 실험 결과 분석

5.1 비용 계산

파일시스템에서 FTL로 보내는 I/O는 읽기 혹은 쓰기이다. FTL은 읽기에 대해서는 매핑을 통한 최신 데이타를 찾아 섹터를 반환하는 단순한 연산이다. 하지만 쓰기는 FTL에서 매핑에 관련된 연산과 플래시 메모리에 대한 읽기와 쓰기, 소거 연산을 유발한다. 따라서 비용은 파일시스템에서 FTL로 보내지는 쓰기에 대해 발생하는 추가비용을 계산하여 각 FTL의 비용으로 한다. 플래시 메모리의 읽기와 쓰기, 소거에 대해 상대적인 비용을 다음과 같이 표기한다.

C_r = read cost of flash memory

C_w = write cost of flash memory

C_e = erase cost of flash memory

FTL의 실험 결과로 얻은 플래시 메모리의 읽기와 쓰기, 소거의 횟수를 파일시스템에서 FTL로 보내지는 쓰기횟수로 나누어 평균을 구한다. 특히 쓰기 횟수는 파일시스템에서 FTL로 보내지는 쓰기횟수를 제외하여 계산한다. 따라서 파일시스템에서 FTL로 보내지는 쓰기 횟수에 대해 추가적으로 발생하는 플래시 메모리의 읽기와 쓰기, 소거의 횟수에 대한 평균을 구한다.

$$p_1 = \frac{N_{read}}{N_{ftl\ write}}$$

$$p_2 = \frac{N_{write} - N_{ftl\ write}}{N_{ftl\ write}}$$

$$p_3 = \frac{N_{erase}}{N_{ftl\ write}}$$

FTL에서 받는 한 번의 쓰기에 대한 읽기와 쓰기, 소거의 평균 횟수인 p_1, p_2, p_3에 앞에서 정한 플래시 메모리의 상대적인 비용을 곱하여 추가비용을 계산한다. 각 FTL 알고리즘을 비교할 때는 FTL에서 수행해야할 한 번의 쓰기비용에 대해 발생하는 평균 추가 비용을 사용하여 비교한다.

$$C_a = p_1 \cdot C_r + p_2 \cdot C_w + p_3 \cdot C_e$$

$$Cost = \frac{C_a}{C_w}$$

5.2 실험 1의 결과 분석

각 알고리즘은 M-system(FMAX), Mitsubishi(MITS), 로그블록기법(BAST), 개선된 로그블록기법(FAST)으로 정한다. 첫 번째 실험 결과는 윈도우즈 파일시스템인 NTFS와 FAT에서 JPG 및 MP3 파일등을 복사/삭제한 로그를 FTL_APAT에서 실험한 결과이다.

5.2.1 파일시스템 간 비교

우선 FTL이 파일시스템의 변화에 따라 얼마나 영향을 받는지 확인하기 위해 윈도우즈 파일시스템간의 차이를 본다. 윈도우즈에서는 크게 FAT과 NTFS 중 한 가지를 파일시스템으로 사용할 수 있다[16]. NTFS는 FAT보다 파일시스템의 보안성, 무결성 등을 더 향상시키기 위해 더 많은 메타 데이타를 기록하고 있다. 따라서 NTFS를 사용하는 경우에는 메타 데이타에 대한 덮어쓰기가 FAT보다 더 많이 발생한다. 따라서 알고리즘의 메타 데이타에 대한 I/O 때문에 발생하는 덮어쓰기와 소거, 소요시간 차이를 본다.

FMAX는 메타데이타에 대한 덮어쓰기가 발생하면 그 블록의 복사블록에 덮어써진 메타데이타를 기록하고, 같은 메타데이타에 대한 덮어쓰기가 또 발생하면 복사블록을 단순블록에서 복합블록으로 바꾸고 이전에 기록한 값 아래에 다시 기록한다. 복사블록에 빈공간이 없으면 메타데이타의 논리적인 블록과 복사블록은 합병된다. 따라서 메타데이타에 대한 덮어쓰기에 의해 발생되는 소거는 복사블록의 빈 공간만큼 지연된 후에 발생한다. MITS는 메타데이타에 대한 덮어쓰기를 여유 공간에 기록하고 여유 공간에 빈 섹터가 없으면 그 블록을 합병한다. 따라서 메타데이타에 대한 덮어쓰기가 발생하면 여유 공간의 빈 공간만큼 소거가 지연된다.

BAST는 로그블록의 개수가 정해져 있고, 한 개의 논리적인 블록은 한 개의 로그블록을 사용한다. 메타데이타에 대한 I/O가 발생하면 하나의 로그블록을 희생블록으로 선택하여 소거하고, 길이가 짧은 메타데이타를 로그블록에 기록하게 된다. 또, 길이가 짧은 메타데이타를 기록하고 있는 로그블록이 희생블록으로 선택되어 소거될 때 로그블록에 빈 섹터가 많이 있는 채로 소거하게 되므로 BAST에서 로그블록을 효율적으로 사용하지 못한다. FAST는 메타데이타에 대한 덮어쓰기가 발생하면 거의 모든 경우가 임의적인 섹터에 대한 덮어쓰기이므로 임의쓰기용 로그블록에 기록된다. 하지만 임의쓰기용 로그블록의 크기 역시 제한되어 있고, 원형 큐 형태로 동작하므로 임의쓰기용 로그블록을 모두 사용하면 임의쓰기용 로그블록의 가장 앞쪽 임의쓰기용 로그블록을 지운다. 임의쓰기용 로그블록에는 모든 블록에서 발생하는 임의적인 덮어쓰기에 대한 기록을 하고 있으므로 임의쓰기용 로그블록이 소거될 때는 여러 개의 블록이 소거될 수 있다. 따라서 메타데이타에 대한 덮어쓰기에 의해 발생되는 소거는 임의쓰기용 로그블록의 빈 공간만큼 지연되고, 소거될 때는 한 번에 여러 개의 소거가 발생한다.

비용계산을 통해보면 표 5와 표 6에서처럼 로그블록을 사용하는 경우의 차이가 그렇지 않은 경우보다 크다. 쓰기에 대한 추가비용만을 고려해 보면 FMAX나 MITS는 로그블록을 사용하는 경우와는 반대로 NTFS일 때 추가비용이 더 적다. FAT은 메타데이타가 모여 있는 섹터에 대해 집중적으로 덮어쓰기가 발생하는 반면, NTFS는 FAT보다 메타데이타가 좀 더 산재해 있기 때문에 발생하는 현상이다. 일정한 섹터가 자주 덮어쓰기 되면 FMAX나 MITS는 블록의 크기 혹은 블록의 절반 크기만을 섹터에 대해 지연하기 때문에 좀 더 적게 쓰기가 발생된다. 산재한 메타데이타에 대한 덮어쓰기에 대해 로그블록 기법은 로그블록의 빈 섹터를 완전히 채우지 못하고 희생 블록으로 선정되어 소거되고, 개선된 로그블록 기법은 다른 블록에 연관된 소거가 발생하여 성능 저하가 심해진다.

표 5 FAT파일시스템상의 평균 추가비용 계산표

FAT	p_1	p_2	p_3	C_a	C_a/C_w
FMAX	4.042	1.023	0.031	17.357	1.736
MITS	1.965	0.850	0.057	16.133	1.613
BAST	1.090	0.090	0.033	5.325	0.532
FAST	0.213	0.156	0.034	5.221	0.522

표 6 NTFS파일시스템상의 평균 추가비용 계산표

NTFS	p_1	p_2	p_3	C_a	C_a/C_w
FMAX	4.045	1.010	0.031	17.207	1.721
MITS	1.963	0.847	0.056	16.082	1.608
BAST	1.136	0.136	0.036	6.062	0.606
FAST	0.266	0.205	0.036	5.911	0.591

표 7 JPG파일의 복사/삭제에 대한 평균 추가비용 계산표

JPG	p_1	p_2	p_3	C_a	C_a/C_w
FMAX	4.202	1.050	0.032	17.873	1.787
MITS	1.966	0.848	0.057	16.093	1.609
BAST	1.195	0.195	0.038	6.989	0.699
FAST	0.374	0.312	0.039	7.423	0.742

표 8 MP3파일의 복사/삭제에 대한 평균 추가비용 계산표

MP3	p_1	p_2	p_3	C_a	C_a/C_w
FMAX	3.905	0.983	0.030	16.723	1.672
MITS	1.961	0.848	0.057	16.097	1.610
BAST	1.043	0.043	0.031	4.574	0.457
FAST	0.111	0.054	0.031	3.776	0.378

따라서 두 배 크기의 물리적인 플래시 메모리 크기를 가져야하는 앞의 두 알고리즘은 전체 크기만큼 임의적인 I/O가 분산되지만 로그블록을 사용하는 두 알고리즘은 로그블록의 수만큼만 분산된다.

5.2.2 복사/삭제한 파일의 크기에 대한 비교

I/O패턴에 따른 차이를 보기위해 실험에서는 JPG 파일을 복사/삭제한 로그와 MP3 파일을 복사/삭제한 로그에 의한 차이를 본다. 각 로그를 분석해 보면 JPG파일을 복사/삭제한 로그는 순차적인 I/O가 많이 발생한 가운데 임의적인 I/O가 자주 발생하고, MP3파일을 복사/삭제한 로그는 순차적인 I/O가 아주 많이 발생한다. I/O는 읽기와 쓰기 구분자와 시작 섹터번호, 섹터의 수 기록하는데, JPG와 MP3을 복사/삭제한 로그에는 주로 128개의 섹터수를 요구하는 I/O가 많았다. 또, 128개의 섹터수를 요구하는 I/O가 발생했을 때 대부분의 경우 시작섹터가 논리적인 블록의 중간지점의 섹터를 가지게 되어 5개의 블록에 대한 순차적인 I/O를 요구하게 된다.

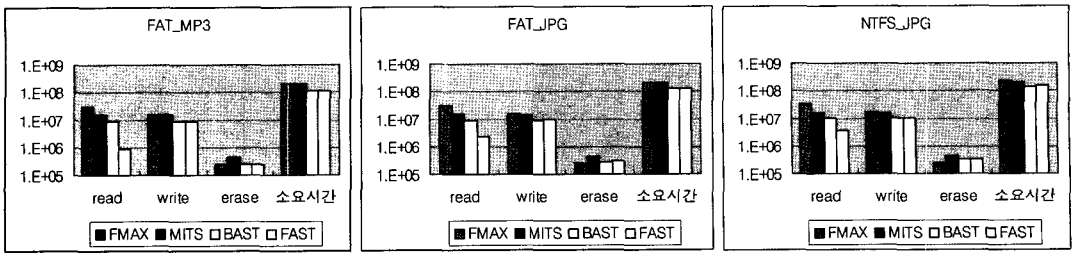

그림 10 실험 1의 결과

따라서 위의 그래프를 비교하면 순차적인 I/O와 임의적인 I/O에 대해 각 알고리즘의 성능을 분석 할 수 있다.

그림 10의 결과를 보면 로그블록을 사용하는 알고리즘의 차이가 더 크다. 이는 파일시스템에서와 마찬가지로 로그블록을 사용하는 알고리즘은 두 배 크기의 물리적인 플래시 메모리가 주어지더라도 논리적인 플래시 메모리 크기와 로그블록으로 사용하는 크기의 합만큼을 물리적인 플래시 메모리에서 사용하기 때문에 발생하는 현상이다. FMAX와 MITS는 논리적인 크기의 플래시 메모리 크기의 두 배 크기를 모두 사용하기 때문에 I/O의 양이 많아지더라도 논리적인 블록의 덮어쓰기의 처리를 물리적인 플래시 메모리의 빈 공간에 기록해둘 수 있다. 따라서 남아있는 플래시 메모리의 공간을 모두 활용하지 않는다는 점은 로그블록을 사용하는 알고리즘의 단점일 수 있다. 하지만 논리적인 플래시 메모리 크기와 물리적인 플래시 메모리의 크기 차이가 거의 없는 것은 로그블록을 사용하는 알고리즘의 강점이다. 실제 경제적인 면에서 이는 매우 중요하다.

비용계산을 통해 각 알고리즘을 분석해 보면, 순차적인 I/O에 대해 교환연산으로 대응할 수 있는 알고리즘인 FMAX, BAST와 FAST 모두 성능이 향상 된다.

5.2.3 실험 1의 결과

그림 11은 실험 1에서의 모든 결과에 대한 평균과 표준편차이다. 표준편차는 I/O에 대해 소요시간이 거의 일정하게 동작하여 사용 중 갑자기 느려지는 현상이 얼마나 발생하는 가에 대해 분석 할 수 있다.

평균그래프를 보면 하나의 I/O요구가 왔을 때 로그블록을 사용하는 알고리즘이 평균적으로 더 빠르게 I/O를 수행한다. FMAX와 MITS는 덮어쓰기를 수행하는 I/O에 대해 데이타블록을 읽고, 덮어쓰기임을 확인한 뒤 복사블록이나 여유 공간에 기록한다. 또, 덮어쓰기의 경우 이전의 데이타 섹터에 대해 사용 불가능(INVALID)함을 기록한다. 따라서 두 알고리즘이 로그블록 기법을 사용하는 알고리즘에 비해 읽기와 쓰기에서 평균 두 배 이상 차이가 난다. 소거부분에서 MITS는 항상 합병을 수행해야하고 한 개의 논리적인 블록에 대해 두 개의 물리적인 블록이 할당되므로 한 개의 논리적인 블록의 덮어쓰기에 대해 소거하는 물리적인 블록은 두 개다. FMAX와 BAST, FAST는 합병연산을 할 때는 한 개의 논리적인 블록의 덮어쓰기에 대해 두 개의 물리적인 블록을 소거하지만, 교환연산을 할 때는 한 개의 물리적인 블록을 소거하므로 소거횟수를 줄인다. 또 읽기횟수에서 FMAX는 복사블록을 찾기 위해 거의 모든 경우에 0번째 섹터를 한 번 더 읽기 때문에 다른 알고리즘의 두 배의 섹터를 읽는다. 그리고 그만큼 쓰기횟수도 0번째 섹터를 유지하기 위해 추가 발생된다.

BAST보다는 FAST가 더 빠르게 동작하는데 쓰기횟수가 더 많더라도 하이브리드 매핑 및 순차쓰기용 로그블록의 현재 사용위치 등을 메모리에서 관리하므로 읽기횟수에서 FAST가 BAST보다 많이 줄어들어 소요시

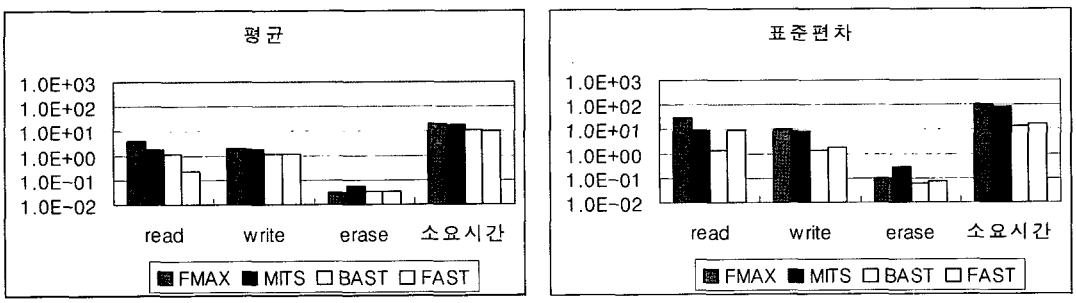

그림 11 실험 1의 결과 평균 및 표준편차

표 9 실험 1의 평균 추가비용 계산표

실험 1 평균	p_1	p_2	p_3	C_a	C_a/C_w
FMAX	4.043	1.016	0.031	17.282	1.728
MITS	1.964	0.849	0.057	16.107	1.611
BAST	1.113	0.113	0.034	5.693	0.569
FAST	0.240	0.181	0.035	5.566	0.557

표 10 실험 2의 평균 추가비용 계산표

실험 2 평균	시간비	p_1	p_2	p_3	C_a	C_a/C_w
FMAX	1.508	5.339	0.767	0.024	15.417	1.542
MITS	1.288	1.488	0.306	0.023	6.894	0.689
BAST	1.092	1.176	0.146	0.020	4.604	0.460
FAST	1.000	0.690	0.118	0.016	3.505	0.350

간 측면에서 좀 더 빠르게 동작한다. FAST에서 순차쓰기용 로그블록을 좀 더 효율적으로 쓰기위해 램 테이블에 몇 가지 정보를 기록한다. FAST는 순차쓰기용 로그블록을 한 개로 두기 때문에 현재 순차쓰기용 로그블록에 기록된 논리적인 블록 번호와 섹터의 기록여부 테이블 등을 램 테이블에 가진다. 따라서 순차쓰기용 로그블록에 기록을 하는 경우 램 테이블을 참조하여 읽기를 수행하지 않고 쓰기를 할 수 있다. 따라서 FAST는 BAST처럼 섹터를 쓰기위해 한 번 읽어야 하는 경우를 많이 줄여서 평균 읽기를 줄여 소요시간이 BAST보다 좀 더 짧다.

하지만 각 I/O에 대한 표준편차를 구하면 BAST가 가장 좋고 FAST가 다음이다. 표준편차가 커지면 한 개의 I/O가 발생했을 때 갑자기 느려지는 현상이 발생할 수 있다. MITS는 논리적인 한 개의 블록에 대해 두 개의 블록을 할당하여 합병을 하므로 길이가 128일 I/O가 발생했을 경우 논리적으로 4~5개의 블록이 합병되어야 한다. 따라서 물리적으로 8~9개의 블록을 합병하게 되므로 그만큼 표준편차는 커지게 된다. 하지만 교환연산이 가능한 나머지 세 알고리즘은 더 적은 수의 블록을 소거하여 편차가 작다. FMAX에서 교환하는 경우는 읽기가 거의 발생하지 않지만 합병하는 경우는 최신에 해당하는 섹터를 찾아 쓰기를 수행해야 하므로 읽기 횟수가 많다. 따라서 FMAX의 소요시간 표준편차도 MITS만큼 크다.

FAST는 임의쓰기용 로그블록이 소거되는 경우 많은 양의 읽기와 쓰기, 소거를 유발하게 된다. 비록 평균 소요시간이 짧지만 임의쓰기용 로그블록이 소거되는 순간 여러 개의 블록이 소거되므로 갑자기 느려지는 현상은 피할 수 없다. 그에 비해 BAST는 합병연산이 한 개의 블록과 한 개의 로그블록에 한정되어 있으므로 표준편차가 가장 작다.

결론적으로 로그블록 기법을 사용하는 것이 평균 소요시간이나 소요시간 표준편차가 가장 작다. 그 중 평균은 임의쓰기와 순차쓰기를 모두 구분하여 그에 알맞게 대응하는 FAST가 가장 빠르고, 표준편차에서는 항상 데이타블록과 데이타 블록에 관련된 로그블록만 소거하는 BAST가 가장 짧은 표준편차를 가진다.

위의 실험 내용을 모두 평균하여 추가비용을 계산해 보면 한번의 FTL에 대한 쓰기에 대한 추가비용은 FAST<BAST<<MITS<FMAX이다.

5.3 실험 2의 결과 분석

실험 2의 플래시 메모리는 읽기 : 쓰기 : 소거의 실제 연산이 10μs:200μs:2000μs으로 지정되었으므로 소요시간

표 11 각 알고리즘별 장단점

알고리즘	장점	단점
FMAX	· 복사블록을 이용하여 덮어쓰기에 대해 단순블록상태->복합블록상태로 바꾸어가며 순차적 쓰기 연산과 임의적 쓰기 연산에 대해 잘 대응함.	· 복사블록을 모든 데이타 블록마다 가져야하므로 논리적으로 보여주는 크기보다 두 배 큰 플래시 메모리를 요구. · 특정섹터에 복사블록의 위치를 기록해야하는 읽기와 쓰기 추가비용.
MITS	· 한 블록 내에 여유 공간을 두어 블록 내에서 발생하는 임의 쓰기 연산에 잘 대응함.	· 여유 공간의 정도에 따라 더 큰 플래시 메모리를 요구. · 한 블록은 교환하지 못하고 항상 합병해야함.
로그블록 기법	· 로그블록에 한 블록에 해당하는 덮어쓰기를 변동섹터 방식으로 기록하여 한 블록에 일어나는 임의 쓰기 연산을 잘 대응함.	· 로그블록의 수에 따라 성능이 정해짐. · 로그블록의 섹터들을 다 채우지 못하고 합병되어 로그블록의 활용성이 낮아짐.
FAST	· 순차쓰기 블록과 임의쓰기 블록으로 나누어 순차적 쓰기 연산과 임의적 쓰기 연산에 맞추어 잘 대응함.	· 순차쓰기 로그블록이 하나만 존재. · 성능 향상을 위해 많은 양의 램 테이블을 사용.

을 위의 비율로 계산한다. 실제 측정한 시간은 각 알고리즘의 연산 시간을 포함한 총 수행시간을 FAST를 기준으로 비율로 나타낸다. 이 실험에 사용된 로그파일은 NTFS에서 JPG와 MP3을 혼합하고 멀티프로세스로 작성한 파일과 리눅스, 디지털카메라, 심비안 등에서 추출한 로그를 이용하여 실험하였다.

실험 2의 결과는 실험 1의 결과에 비해 성능의 순서는 차이가 없으나 각 알고리즘간의 격차는 더 커졌다. 실험 2에서 사용한 로그는 단일 프로세스로 수행한 로그이고 I/O의 횟수가 적으므로 FTL 알고리즘은 실험 1에서의 결과에 비해 더 부정확한 결과를 가진다. 이는 모든 블록에 대해 소거가 여러 번 발생할 만큼의 많은 양의 I/O를 포함하고 있지 않기 때문이다. 하지만 추가비용을 통해 가장 좋은 성능을 가지는 알고리즘은 FAST이다. 시간비는 각 알고리즘이 동작하는 연산 시간이 포함되므로 좀 더 많은 램테이블을 사용하고 복잡한 알고리즘을 가지는 로그블록 기법은 더 많은 연상을 필요로 한다. 따라서 추가비용의 격차보다 시간비의 격차는 더 줄어들지만 순서의 변화는 없다.

5.4 종합분석

각 알고리즘의 전체적인 성능은 전체 소거횟수, 전체 소요시간 과 평균 소거 횟수, 평균 소요시간, 각각의 표준편차 등에 비교해 봤을 때 FAST가 가장 좋다. 이는 하이브리드 매핑을 통해서 순차적인 덮어쓰기와 임의적인 덮어쓰기에 대해 가장 효과적으로 대응하기 때문이다.

플래시 메모리에 사용할 파일 시스템은 각 알고리즘의 성능에 크게 영향을 주지 않지만 NTFS를 쓰면 전체 I/O양이 늘어나고 임의적인 I/O가 많이 발생하므로 가능한 적은 양의 I/O를 위해 FAT을 쓰는 것이 좋다. 또, 같은 양의 파일을 복사/삭제 할 경우 파일 크기가 크면 파일의 크기만큼 순차적인 쓰기/덮어쓰기가 발생하므로 작은 파일을 자주 복사/삭제 하여 임의적인 I/O를 증가 시키는 경우보다 더 적은 소거횟수를 가진다.

마지막으로 각 알고리즘은 스스로의 특징을 활용하여 CPU의 연산속도를 줄이거나 사용할 램 테이블의 크기를 줄이면서 좋은 성능을 내는 FTL을 추구하고 있다. 하지만 블록매핑만 하게 되면 블록내의 섹터를 찾기 위해 추가적인 정보를 저장하거나 읽기와 쓰기 횟수를 증가해야한다. 따라서 하이브리드 매핑을 통해 플래시 메모리의 읽기와 쓰기 횟수를 줄이면서 좀 더 작은 양의 물리적인 플래시 메모리를 사용하는 로그블록기법을 적용하는 것이 경제적으로 좋다. 또, FAST와 같이 파일 시스템에서 주로 발생하는 I/O패턴을 예측하여 그에 알맞게 동작하는 FTL이 좋은 성능을 낸다.

모든 알고리즘들은 아직도 좀 더 좋은 성능을 위해 개선해야 할 점이 존재한다. FMAX는 복사블록을 찾거나 복사블록의 빈 섹터를 찾기 위해 발생하는 읽기횟수를 줄일 수 있는 방법을 찾아야한다. MITS는 교환연산과 유사하게 소거를 줄일 수 있는 방법을 모색해야하고, BAST는 로그블록의 활용성을 좀 더 높일 수 있는 방법을 찾아야한다. FAST는 각 패턴에 맞게 대응하기위해 발생하는 CPU 비용을 줄이고 순차쓰기용 로그블록의 수를 증가해 로그블록의 활용성을 높일 수 있는 방법을 찾아야한다. 그리고 모든 알고리즘은 앞으로 대용량 플래시 메모리가 등장하면서 디스크 대체용으로 플래시 메모리가 사용되었을 때에 발생할 수 있는 대용량화, 고속화에 대한 해결방안을 모색해야 한다.

6. 결 론

최근 다양한 휴대용 장치가 등장함에 따라 플래시 메모리가 주목받고 있다. 그동안 보조기억장치로 자리매김을 해왔던 디스크보다 장점이 많기 때문이다. 하지만, 플래시 메모리를 효율적으로 사용하기 위해서는 FTL이 필수이다. 플래시 메모리 저장 장치에서 여러 FTL 알고리즘을 평가하기 위해서는 유용한 디스크 I/O 패턴을 얻어야 한다. 본 논문에서는 현재 가장 많이 사용되고 있는 윈도우즈 운영체제의 I/O 패턴을 얻기 위해 디스크 드라이버의 필터 드라이버를 작성하여 디스크 I/O 패턴을 얻었으며, 이를 실시간으로 FTL 알고리즘에 적용해 보며 성능을 평가할 수 있는 FTL APAT를 개발하였다.

FTL APAT를 통해 FTL 알고리즘을 분석한 결과에서 가장 좋은 성능을 보이는 알고리즘은 FAST이다. FAST는 제한된 크기의 로그블록을 가진다는 점을 보완 한다면 좀 더 나은 알고리즘으로 발전 할 수 있다. 또 순차쓰기용 로그블록의 활용성을 높이고 CPU비용을 낮추는 점도 보완해야할 과제이다. BAST는 로그 블록을 좀 더 활용성 높게 사용할 수 있는 방안을 찾아야 한다. 그 외의 다른 알고리즘 들은 여러 가지 I/O 패턴에 대해 효과적으로 대처 할 수 있도록 해야 하고, 논리적으로 제공되는 플래시 메모리의 양보다 더 많은 양의 물리적인 플래시 메모리를 제공해야 하는 점을 보완해야 한다.

실험에 사용한 알고리즘들은 I/O가 시작섹터번호와 섹터의 개수를 인자로 들어오면 하나의 섹터씩 나누어 수행하고 있다. 디스크 대체용 플래시 메모리는 디스크처럼 클러스터 단위를 고려하거나 섹터 개수를 고려하여 I/O를 수행하는 방법이 추가되어야 한다. 또 플래시 메모리의 대용량화에 따른 변화도 필수이다.

마지막으로 플래시 메모리의 수명과 관계되는 소거연산 평준화(wear leveling)[17]를 각 알고리즘 별로 잘 구현하여 각 블록의 평균 수명을 일정하게 유지하고 데

이타 전송량을 증가하면 플래시 메모리 저장 장치가 디스크를 대체하였을 경우 좋은 성능을 보여줄 것이다.

참 고 자 료

[1] BAN, A. 1995. Flash file system. US patent 5,404,485. Filed March 8, 1993; Issued April 4,1995; Assigned to M-Systems.

[2] INTEL CORPORATION. 1998b. Understanding the flash translation layer(FTL) specification. Application Note 648, Intel Corporation.

[3] E. Gal and S. Toledo. Algorithms and data structures for flash memories. ACM Computing Surveys, 37(2):138-163, 2005.

[4] Bum-soo Kim, Gui-young Lee, Method of driving remapping in flash memory and flash memory architecture suitable therefor, 2002. US Patent, no. 6,381,176 B1.

[5] Petro Estakhri, Berhanu Iman, Moving sequential sectors within a block of information in a flash memory mass storage architecture, 1999. US Patent, no. 5,930,815.

[6] Takayuki Shinohara. Flash memory card with block memory address arrangement, 1999. US Patent, no. 5,905,993.

[7] Amir Ban. Flash file system optimized for page-mode flash technologies, 1999. US Patent, no. 5,937,425.

[8] Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min, and Yookun Cho. A space-efficient flash translation layer for compact flash systems. IEEE Transactions on Consumer Electronics, 48(2), 2002.

[9] Sang-Won Lee and Dong-Joo Park. FAST:An efficient flash translation layer for flash memory, Submitted for publication, 2005.

[10] Rosenblum, M. and Ousterhout, J. K. 1992. The design and implementation of a log-structured file system. ACM Transactions on Computer Systems 10, 1, 26-.52.

[11] Axis Communications. 2004. JFFS home page. Electronic document available online at http://developer.axis.com/software/jffs/. Lund, Sweden.

[12] Woodhouse, D. 2001. JFFS: The journaling flash file system. Presented in the Ottawa Linux Symposium, July 2001 (no proceedings); a 12-page article is available online at http://sources.redhat.com/jffs2/jffs2.pdf.

[13] Aleph One. 2002. YAFFS: Yet another flash filing system. Electronic document available online at http://www.aleph1.co.uk/yaffs/index.html. Cambridge, UK.

[14] Art Baker, Jerry Lozano, "Windows 2000 Device Driver Book : A Guide for Programmers 2nd Edition," Prentice Hall, 2000.

[15] Disk Subsystem Performance Analysis for Windows. Electronic document available online www.microsoft.com/whdc/device/storage/subsys_perf.mspx

[16] O'Reilly, "Windows NT File System Internals, Develo- per's Guide," Rajeev Nagar

[17] Wells, S. E. 1994. Method for wear leveling in a flash EEPROM memory. US patent 5,341,339. Filed November 1, 1993; Issued August 23, 1994; Assigned to Intel.

박 원 주

2006년 한국외국어대학교 컴퓨터및정보통신공학과 학사. 2003년~현재 한국외국어대학교 컴퓨터및정보통신공학과 석사 관심분야는 플래시메모리, 데이타베이스, 임베디드 시스템

박 성 환

2006년 한국외국어대학교 컴퓨터및정보통신공학과 학사. 2003년~현재 한국외국어대학교 컴퓨터및정보통신공학과 석사 관심분야는 플래시메모리, 데이타베이스, 임베디드 시스템

박 상 원

1994년 서울대학교 컴퓨터공학과 학사 1997년 서울대학교 컴퓨터공학과 석사 2002년 서울대학교 컴퓨터공학과 박사 2002년~2003년 세종사이버대학교 디지털콘텐츠학과 전임강사. 2003년~현재 한국외국어대학교 정보통신공학과 조교수 관심분야는 플래시메모리, 임베디드 데이타베이스, XML