Locality를 이용한 섹터 매핑 기법의 효율적인 FTL 알고리듬

준회원 홍 수 진*, 정회원 황 선 영*

Design of an Efficient FTL Algorithm Exploiting Locality Based on Sector-level Mapping

Soo-Jin Hong* *Associate Member*, Sun-Young Hwang* *Regular Member*

요 약

본 논문은 플래쉬 메모리 접근 시 소거 횟수를 줄이기 위해서 섹터 매핑 기법을 바탕으로 지역성(Locality)을 적용한 FTL (Flash Translation Layer) 알고리듬을 제안한다. 섹터 매핑 기법은 다른 기존의 매핑 기법보다 성능은 우수하나, 매핑 테이블이 커지는 단점을 가진다. 제안한 알고리듬에서는 동적 테이블 업데이트 방식으로 매핑 테이블의 크기를 줄였으며, 순차적 지역성(Sequential Locality)를 이용하여 순차쓰기를 처리하고 임의쓰기에서 hot 섹터를 추출하였다. 실험결과 제안된 알고리듬은 기존의 BAST, FAST, 섹터 매핑 알고리듬과 비교하여 75.2%, 65.8%, 10.3%의 소거 횟수가 감소함을 보인다.

Key Words : Flash memory, FTL, Sector-level mapping, Embedded system, File system

ABSTRACT

This paper proposes a novel FTL (Flash Translation Layer) algorithm employing sector-level mapping technique based on locality to reduce the number of erase operations in flash memory accesses. Sector-level mapping technique shows higher performance than other mapping techniques, even if it requires a large mapping table. The proposed algorithm reduces the size of mapping table by employing dynamic table update, processes sequential writes by exploiting sequential locality and extracts hot sector in random writes.

Experimental results show that the number of erase operations has been reduced by 75.4%, 65.8%, and 10.3% respectively when compared with well-known BAST, FAST and sector mapping algorithms.

I. 서 론

최근 모바일 기기의 급속한 발달과 함께 모바일 기기의 저장장치도 함께 발달하고 있다. 지금까지 모바일 기기의 저장장치는 주로 광학 메모리(HDD)를 사용해왔으나, 광학 메모리는 기계 구동방식으로 전반적인 시스템의 반응성이 느리고 구동부가 있기 때문에 견고성이 떨어질 뿐만 아니라 전력 소모량이 많고 충격에 약하며 소음 및 발열이 많으므로 휴대성이 떨어진다. 차세대 저장매체인 NAND 플래시 메모리는 모바일 저장장치로서 적합한 특징을 갖고 있다. 전력 소모와 발열이 적고 access time이 빠르며 외부 충격에 강하고 면적이 작으므로 휴대성이 좋다. 모바일 저장장치로서 플래쉬 메모리에 대한 수요가 증가하고 있으며, 차세대 저장매체인 NAND 플래쉬 메모리에 기반한 반도체 디스크(Solid State Disk)가 일반 PC의 하드 디스크를 대체하고 있다. 특히 SSD는 전력소모와 면적에 민감한 노트북 분야에서 활발한 움직임을

※ 본 연구는 교육과학기술부의 재원으로 한국연구재단의 지원에 의해 수행되었습니다.(#2010-0008043).
* 서강대학교 전자공학과 CAD & ES 연구실 (hwang@sogang.ac.kr)
논문번호 : KICS2011-05-213, 접수일자 : 2011년 5월 11일, 최종논문접수일자 : 2011년 6월 23일

보이고 있다[1,2]. 최근 SSD의 사용이 급격이 늘어나고 있는 추세이나, 모바일 기기에서는 플래쉬 메모리의 사용이 여전히 채택되고 있다. 본 논문의 목적은 모바일 기기에 사용되는 플래쉬 메모리를 타겟으로 하여 이를 효율적으로 컨트롤 함으로써 플래쉬 메모리의 성능을 향상시키는데 있다.

플래쉬 메모리는 몇 가지 제약 사항이 있다. 첫 번째 문제점은 erase-before-write 문제이다. 플래쉬 메모리에서 소거 동작의 연산 단위는 블록이고 읽기/쓰기 동작의 연산 단위는 섹터로서, 연산 단위의 차이에 의해서 문제가 발생한다.[3] 플래쉬 메모리는 덮어쓰기가 불가능하기 때문에 기존의 데이터를 소거한 후 쓰기를 해야 하나 소거 동작은 블록 단위 연산이므로 같은 블록 내 다른 섹터들도 소거된다. 덮어쓰기의 경우 변경된 데이터를 저장할 새로운 공간을 찾아 확보하고 변경된 데이터를 새로운 공간에 저장하고 기존의 데이터를 소거한다. 두 번째 문제점은 읽기, 소거, 소거의 연산속도가 다르다는 점이다. 삼성 K9WBG08U1M NAND 플래쉬 메모리의 경우 읽기 동작에 걸리는 시간은 25us, 쓰기 동작에 걸리는 시간은 200us, 소거 동작에 걸리는 시간은 2ms로서 읽기/쓰기 동작에 비하여 소거 동작에 소요되는 시간이 길다.[3] 전체 동작 속도를 줄이기 위해서는 소거 동작 횟수를 최대한 줄여야 한다. 세 번째로 플래쉬 메모리는 블록당 10만~100만 번의 소거 횟수 제한이 있으며 일정 횟수를 넘어가면 오작동 문제가 발생하는 경우가 있다. 이 문제점들은 소거 동작과 관련이 있으며 플래쉬 메모리의 성능을 향상시키기 위해서는 소거 동작 횟수를 최소화시켜야 한다.

이와 같은 문제점들을 해결하기 위하여 FTL (Flash Translation Layer)의 역할이 중요하다. 그림 1은 플래쉬 메모리 시스템의 일반적 내부 구조를 보인다. 파일 시스템에서 처리하는 논리적인 주소에 대한 요청이 들어오면 FTL은 논리 주소를 플래쉬 메모리의 물리적인 주소로 변환하는 역할을 하며, 부가적으로 wear-leveling 기능을 제공하고 전원 오류의 상황에서 데이터의 안정성을 보장해 준다. 여기서, 적용하는 FTL 알고리듬에 따라 연산 횟수가 달라지므로 FTL은 플래쉬 메모리의 성능을 좌우한다[4].

FTL의 어드레스 매핑 기법에는 섹터 매핑 기법, 블록 매핑 기법, 하이브리드 매핑 기법이 제안되었다[2,5-7]. 섹터 매핑 기법은 섹터를 단위로 논리 주소와 물리 주소가 일대일로 매핑되며, flexibility와 성능이 가장 좋지만 mapping table이 너무 커지는 단점이 있다[6]. 블록 매핑 기법은 논리적 블록이 1:1로 물리적

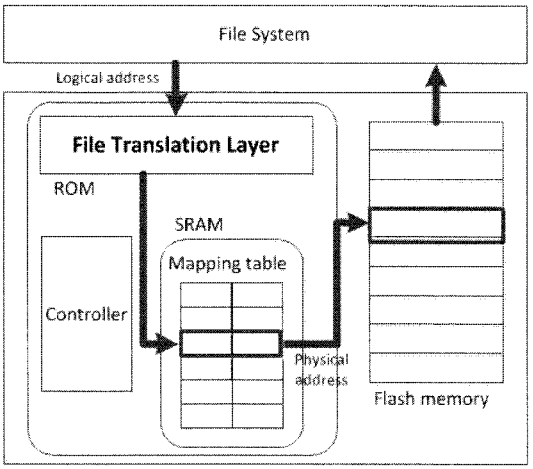


그림 1. 섹터 매핑 테이블과 블록 정보 테이블.

블록에 매핑되어, 이 블록 번호와 offset으로 해당 섹터를 찾기 때문에 mapping table의 크기는 작으나 전체 동작 속도가 느려지는 단점이 있다[7]. 하이브리드 매핑 기법은 섹터 매핑 기법과 블록 매핑 기법을 결합시킨 것으로 cache와 같은 역할을 하는 로그 블록에는 섹터 매핑 기법을 적용시키고 데이터 블록에는 블록 매핑 기법을 이용한다[2].

모바일 기기의 주 활용 대상 중의 하나가 멀티미디어 어플리케이션 재생이다. 동영상 실시간 재생이나 음악 재생 등은 주로 파일을 처음부터 끝까지 순차적으로 읽는 동작이므로 데이터가 순차적으로 접근되는 경향성이 있다. 그러나 처음부터 끝까지 순차적 접근만 있는 것이 아니며 순차 접근과 임의 접근이 혼합되며, 동영상 재생과 같은 멀티미디어 재생의 경우는 순차적 지역성(sequential locality)이 높다는 특징을 이용하여 순차 접근과 임의 접근을 따로 처리하여 효율성을 높일 수 있다. 임의 접근의 경향성을 살펴보면 데이터 크기는 작지만 빈번하게 접근되는 데이터가 존재하며, 이를 Hot data라고 한다. 이 값들은 무효화 값들을 많이 만들어내기 때문에 제대로 처리를 해주지 않으면 메모리의 성능을 크게 떨어뜨릴 염려가 있다[8].

본 논문에서 모바일 기기에서 사용되는 플래쉬 메모리로 상황을 가정하여 연구를 수행하였다. 플래쉬 메모리가 주로 사용되는 환경이 모바일 환경이며 모바일 환경은 일반 PC와는 사용되는 패턴이 다르다. 모바일 환경에서는 지역성이 높은 일들을 많이 처리하므로 그 일을 처리하는데 특화된 알고리듬을 제안하였으며, 섹터 매핑 기법을 적용 시 매핑 테이블이 커지는 단점을 보완하기 위해서 매핑 테이블 업데이

트 방식과 동적 할당 기법을 이용하였다.

본 논문의 구성은 다음과 같다. 2절에서는 기존의 제안된 FTL의 특징을 설명하고, 3절에서는 본 논문에서 제안된 FTL의 구조와 알고리듬을 설명한다. 4절에서는 각각의 어플리케이션마다 소거 횟수를 측정한 실험 결과를 보인다. 마지막으로 5절에서는 결론을 제시한다.

Ⅱ. 관련 연구

플래쉬 메모리에서 덮어쓰기가 지원되지 않는 물리적 제약을 해결하기 위하여 FTL에 대한 연구가 활발히 진행되었다. 덮어쓰기 동작을 구현하는 가장 간단한 방법은 해당 섹터가 속한 블록을 소거하고 다시 쓰는 방법이다. 이 방법은 섹터가 update 될 때마다 copy와 소거동작을 반복해야 하기 때문에 성능 면이나 NAND 플래쉬의 수명 측면에서 바람직하지 않다. 이러한 문제점을 해결하기 위해서 BAST는 cache개념의 로그 블록을 두어서 덮어쓰기 동작 시 해당 섹터만을 로그 블록에 쓴다[2]. 데이터 블록과 로그 블록은 1:1 관계이며, 파일 시스템의 쓰기 패턴에 따라서 병합 방법이 다르다. 그로 인해 순차 쓰기의 비중이 높은 곳에서는 좋은 성능을 발휘하지만 임의 쓰기에서는 낮은 성능을 보인다. 또한 block - thrashing 이라는 문제점이 발생한다. BAST와 마찬가지로 FAST도 로그 블록을 사용하지만 BAST와는 다르게 데이터 블록과 로그 블록을 1:N mapping 으로 확장시켰다[9]. 이 경우 로그 블록의 공간 이용도가 높아지지만 소량의 임의 쓰기가 많을 경우 성능이 많이 떨어지게 된다.

EAST는 동적 할당 블록 매핑 기법을 이용하여 재할당 블록을 사용하였다[10]. 이 방법은 플래쉬 메모리의 공간을 효율적으로 사용하게 한다. 여기서 재할당 블록이란 사용하지 않는 data 블록을 로그 블록처럼 사용하는 것을 지칭한다.

LAST는 FAST를 계승하는 알고리즘이다[11]. 임의쓰기과 순차 쓰기를 구별하여 로그블록을 할당하였다. 그러나 오로지 데이터 사이즈만으로 구별하기 때문에 작은 크기의 순차 쓰기는 구분하지 못한다. 또한 임의 로그 블록을 다시 hot partition과 cold partition으로 구별하였다. 이 구분을 통하여 garbage collection의 오버헤드를 줄였다.

참고문헌 [12]는 소거 횟수가 적은 섹터 매핑기법을 적용하고, 매핑 테이블을 동적으로 할당하여 테이블의 크기를 줄이는 연구가 진행되었다. 그러나 지역성에 대한 고려가 없어서 garbage collection의 오버헤드가 발생하며, 프로그램에서 많은 LSN(Logical Sector Number)을 사용하면 동적 할당 기법을 이용하여도 매핑 테이블의 크기는 커지는 문제점이 있다.

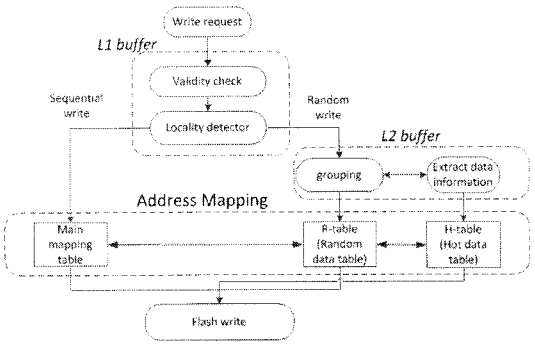


그림 2. 제안한 FTL 알고리듬 동작 흐름도

본 논문은 플래쉬 메모리의 공간 활용도가 높은 섹터 매핑 기법을 바탕으로 순차적 지역성(sequential locality)과 시간적 지역성(temporal locality)을 고려하여 소거 횟수를 줄이고 garbage collection의 오버헤드를 줄였다. 또한 순차 쓰기의 특성을 활용하여 매핑 테이블의 크기를 줄였다.

Ⅲ. 제안한 FTL 알고리듬

본 절에서는 제안한 FTL 알고리듬에 관해 기술한다.

3.1 개 관

제안한 FTL 알고리듬은 지역성을 이용하여 임의쓰기와 순차쓰기를 구분하며, 임의쓰기를 grouping 한 후 빈번하게 접근되는 데이터를 분리하여 관리한다. 제안한 FTL 알고리듬을 설명하기 위해 그림 2에서 전체 동작 흐름도를 보인다. 섹터 S의 쓰기 요청이 발생하면 L1, L2 buffer 또는 플래쉬 메모에 이미 같은 LSN의 데이터가 있는지 확인해보고 있다면 이미 적혀져 있는 데이터를 무효화한다. 이것은 새로운 시퀀스가 들어올 때마다 처리해 주어야 한다. 이후에 임의쓰기와 순차쓰기를 구분하는 작업을 한다. 쓰기 요청은 임의쓰기와 순차쓰기가 복합적으로 이루어진다. SSD가 주로 사용되는 모바일 기기에서는 순차쓰기가 임의쓰기에 비해서 많이 사용되며, 각 쓰기에 따른 특징이 있기 때문에 이것을 구분하여 관리하면 효율성을 높일 수 있다. 임의쓰기와 순차쓰기의 구분 기준은 임의로 정하는 K 섹터이며, 이것은 순차적 지역성을 이용한 것으로 쓰기 요청이 순차적으로 K 섹터 이상 오는 경우에 그 쓰기 시퀀스는 순차쓰기로 분류한다.

만약 K 값을 너무 크게 정하면 순차쓰기로 처리해야 하는 시퀀스가 임의쓰기로 처리되고, K 값을 너무 작게 정하면 순차쓰기가 임의쓰기로 처리되므로 신중하게 선택해야 한다. 이 과정은 L1 buffer에서 진행된다.

순차쓰기의 경우 주소 매핑 과정을 거쳐서 바로 플래쉬 메모리에 데이터를 쓰게 되나 임의쓰기 시퀀스는 L2 buffer에 grouping 되며 이렇게 모인 임의쓰기에서 hot 데이터를 추출한다. Hot 데이터는 크기는 작으나 빈번하게 접근되므로 데이터 갱신이 빈번하게 일어나며 무효화된 섹터가 많이 발생한다. 무효화될 hot 데이터를 같은 블록에 모으면 Garbage collection의 효율성을 높일 수 있다.

주소 매핑 테이블은 크게 주 매핑 테이블과 R-table (Random data table) 그리고 H-table(Hot data table)로 구성된다. 주 매핑 테이블은 Physical Block에 매핑되는 Logical Block에 대한 정보를 가진다. R-table은 임의 쓰기의 논리주소와 물리주소를 매핑하며, H-table은 L2 버퍼에서 추출한 hot 데이터를 기록한다. 주소 매핑 과정 후 플래쉬 메모리에 데이터를 기록한다.

3.2 제안한 매핑 알고리듬의 특징

제안한 FTL 알고리듬은 섹터 매핑 기법을 바탕으로 하며 L1, L2 버퍼를 두어 순차적 지역성과 시간적 지역성을 이용하여 순차쓰기와 임의쓰기를 구분하고 임의쓰기에서 hot data 정보를 추출한다. 이렇게 분류한 데이터는 분류된 특징을 살려서 주 매핑 테이블과 R-table, 그리고 H-table들을 구성한 후 플래쉬 메모리에 기록된다. 또한 주 매핑 테이블의 블록 정보를 이용하여 병합 연산 시 victim 블록을 선정한다. 섹터 매핑 기법은 섹터간 매핑이기 때문에 매핑 테이블이 커지는 단점을 보완하기 위하여 동적 테이블 업데이트 기법을 이용하였다.

3.2.1 L1 버퍼와 L2 버퍼

제안한 알고리듬은 L1 버퍼를 이용하여 순차쓰기와 임의쓰기를 구분하므로 멀티미디어 어플리케이션 실행에 유리하다. L1 버퍼는 8 개 섹터 크기의 메모리로 4 섹터 이상 순차적으로 L1 버퍼를 채우면 순차쓰기라고 가정하였다. 멀티미디어 어플리케이션 재생 시 동작들은 대부분 순차 쓰기와 읽기로 이루어지며 임의 쓰기와 읽기는 발생 빈도가 적으므로 이를 따로 처리하여 관리의 효율성을 높였다. L2 버퍼에서는 8 섹터 안에 반복되는 임의쓰기가 있으면 그 데이터는 빈번한 것이라고 가정하였으며, hot 데이터로써 h flag가 태크되며 따로 H-table을 구성한다. 버퍼는 4 섹터씩 업데이트가 되며, 이때 유효한 데이터들만 플래쉬 메모리에 기록된다. 순차쓰기와 임의쓰기를 구분해서 생기는 부가적인 장점은 테이블 크기를 효과적으로 줄일 수 있다는 점이다. 기존의 섹터 매핑 기법의 테이블은 논리 섹터 번호마다 물리 섹터 번호가 매핑이 되므로 테이블의 열의 개수는 논리 섹터의 수와 같게 된다. 순차쓰기는 연속되는 수이므로 처음에 쓰여지는 논리주소와 물리주소 그리고 순차쓰기가 쓰여진 길이만 알면 나머지 값들은 계산을 통해서 주소를 매핑시킬 수 있으므로 섹터 매핑 기법이지만 블록 단위의 매핑 테이블을 구성할 수 있다. 플래쉬 메모리에 데이터가 도달하기 전에 버퍼들 안에서 무효화된 값들은 플래쉬 메모리에 저장되지 않으므로 그만큼 플래쉬 메모리에서 병합 연산과 소거 횟수가 줄어든다.

3.2.2 Mapping table 의 특징과 동적 할당 기법

주 매핑 테이블은 각각 Physical Block 에 매핑되는 Logical Block에 대한 정보를 포함한다. 섹터 매핑 기법은 섹터를 기본 단위로 하므로 테이블도 섹터단위로 구성되지만, 제안한 알고리듬은 s_LSN(starting LSN), s_PSN(starting PSN), Length field를 이용하여 블록단위 매핑 테이블을 구성한다. 순차쓰기의 특징을 이용하면 시작점과 그 길이 정보만으로 시퀀스를 이루는 데이터들을 알 수 있다. 플래쉬 메모리는 한 블록 당 4 개의 섹터로 이루어져있으며, 쓰기 시퀀스의 길이는 정해져 있지 않다. 순차적으로 LSN이 4 섹터 이상 요청되어야 순차 시퀀스로 보기 때문에, 한 블록 내에 서로 다른 시퀀스가 최대 2개 들어갈 수 있다. 이때 s_LSN에는 Physical block의 첫 섹터와 매핑하는 LSN과 새로운 시퀀스의 시작 LSN, 이렇게 최대 2개의 LSN 값이 기록될 수 있으며 s_PSN은 s_LSN에 매핑되는 물리적 주소이고 Length는 순차 시퀀스의 길이이다. 순차 시퀀스의 길이가 정해져 있지 않기 때문에 블록 내에서 시퀀스가 끝나버릴 수도 있고 다음 블록으로 넘어갈 수도 있다. 순차 시퀀스가 다음 블록으로 넘어가는 경우 Cont/Break PBN (Continue/Break PBN) field에서는 다음 블록주소를 가리키며, (C, next PBN #)으로 표현된다. 시퀀스가 블록 내에서 끝나는 경우 B를 기록하며, 그 다음 시퀀스가 올 경우에는 그 다음 시퀀스로 인해 오는 블록 주소를 기록하고 그렇지 않을 경우에는 비워둔다. 각각 (B, next PBN #), (B,-)으로 표현한다. 주 매핑 테이블의 LSN과 PSN 매핑은 순차쓰기의 경우에만 기록이 가능하므로 임의쓰기 데이터 블록은 주 매핑 테

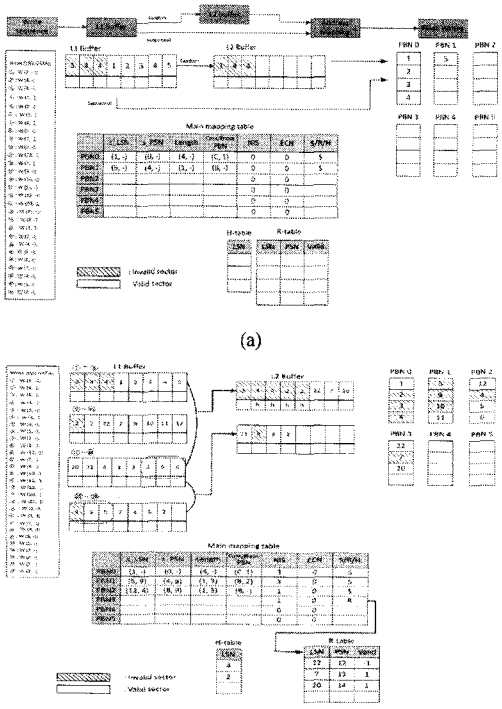


그림 3. 플래쉬 메모리 공간 사용과 쓰기 연산 처리. (a) 쓰기 시퀀스 ①~⑧의 쓰기 연산 처리, (b) 쓰기 시퀀스 ⑨~2*의 쓰기 연산 처리

이블의 S/R/H(sequential data block/random data block/Hot data block)field에서 R로 표시하고 R-table을 통해서 LSN과 PSN을 매핑한다. Hot 데이터 블록은 주 매핑 테이블의 S/R/H field에서 H로 표시하지만 다른 임의쓰기처럼 R-table에 LSN과 PSN을 기록한다. 주 매핑 테이블이 블록 단위로 이루어져있기 때문에 블록 업데이트에 따라서 테이블도 업데이트를 하며, 이 방법은 테이블의 크기가 더 이상 늘어나지 않도록 한정시킨다. 메모리가 다 찰 경우 새로운 데이터를 저장하기 위해서는 병합 연산을 하게 되며 이때 victim 블록을 선정하는데 주 매핑 테이블의 NIS (Number of Invalid Sectors), ECN(Erase Count Number)의 값을 이용한다. NIS은 블록 내의 무효화된 섹터 수이며, 무효화된 섹터가 많을수록 병합 시 데이터 copy 양이 줄어든다. ECN은 블록이 지워진 총 횟수이며 NAND Flash는 소거 횟수가 정해져 있으므로 블록마다 소거 횟수가 균등하도록 wear leveling을 구현해야 하므로 병합 연산 시 victim 블록으로 ECN 횟수가 가장 적은 블록을 선택한다. 즉, victim 블록은 병합 연산 시 NIS가 가장 크고 ECN이 가장 작은 블록이다. 기존의 섹터 매핑 기법은 초기에 모든 LSN과 PSN 에 대한 맵을 가지나, 제안한 알고리듬은 동적 할당 기법을 이용하여 요구된 LSN에 대해서만 매핑 정보를 가지므로 테이블의 크기를 줄일 수 있으며, 모든 매핑 테이블에 동적 할당 기법을 적용하였다.

3.3 쓰기 연산 (Write operation)

본 절에서는 예제를 통하여 제안한 FTL이 파일 시스템에서 쓰기 요청을 처리하는 알고리듬을 설명한다. 그림 3은 제안한 알고리듬의 플래쉬 메모리 공간 사용과 쓰기 시퀀스 ①~2*의 쓰기 연산 처리 과정을 보인다. 그림 3(a)에서 블록 당 섹터의 수는 4이고 전체 블록의 수는 6이다. 파일 시스템에서 쓰기 요청이 왔을 때 처음 L1 버퍼를 거치게 되며 L1 버퍼에서는 쓰기 시퀀스가 들어갈 때마다 무효화 check를 하고 순차쓰기와 임의쓰기를 분류한다. 그림 3(a)에서 LSN 4가 들어온 다음에 또다시 LSN 4가 들어왔으므로 이전 LSN 4는 무효화되며"무효화 표시(-1)"를 한다. ①~②번 시점에서는 연속이지만 4섹터 이하의 순차 시퀀스이므로 임의쓰기이나 ④~⑧번의 시점에서는 연속적인 4섹터 이상의 순차 시퀀스이므로 순차쓰기이다. 임의쓰기는 L2 버퍼로 진행하고 순차쓰기는 바로 플래쉬 메모리에 기록되며 매핑 테이블을 구성한다. PBN 0를 채우는 첫 LSN이 LSN 1이며 시퀀스의 길이가 5이므로 다음 블록으로 시퀀스가 넘어간다. ④~⑧ 시퀀스 외에 새로 시작하는 시퀀스가 없으므로 s_LSN에는 (1,-)이 기록되며 s_PSN에는 이에 대응하는 값이 기록된다. 블록 당 섹터 수가 4이므로 Length의 최대값은 4 sector이며 순차 시퀀스 하나로만 채워진 PBN 0의 Length는 (4,-)로 기록되며 ⑦ 번째 쓰기 요청인 LSN 4까지 기록된다. ⑧ 번째 쓰기 요청인 LSN 5는 다른 블록에 저장되나, 같은 순차 시퀀스이므로 Cont/Break PBN에 LSN 5가 저장된 PBN 1을 기록하며 기록 형태는 (C,1)이다.

그림 3(b)은 쓰기 시퀀스 ⑨~2*의 쓰기 연산 처리 과정을 보인다. 그림 3(b)도 위와 같은 방법으로 쓰기 시퀀스들이 진행된다. L1 버퍼는 쓰기 시퀀스들이 8섹터를 다 채우면 8섹터씩 업데이트가 되며 임의쓰기는 L2 버퍼, 순차쓰기는 플래쉬 메모리로 이동한다. 2@~2$ 시점의 LSN 4,5,6 의 경우를 보면 임의쓰기 섹터나 연속되는 섹터가 2개 혹은 3개인 경우에는 L1 버퍼 업데이트 시 쓰기 종류에 대한 판단이 불분명하다. 뒤따라오는 시퀀스가 LSN 7일 경우 LSN 4,5,6은 순차쓰기에 해당하고 LSN 7 외의 값일 경우 임의쓰기에 해당하기 때문에 LSN 4,5,6만으로는 판단할 수

없으므로 L1 버퍼에서 delay를 시킨다. L2 버퍼는 L1 버퍼에서 임의쓰기로 판단된 데이터들을 8섹터씩 모아둔다. L2 버퍼는 hot 데이터를 추출하기 위해 사용되며, L2 버퍼의 8 섹터가 다 찼을 경우에 hot 데이터 추출 과정을 진행한다. 8 섹터 안에 같은 LSN이 있을 때 그 값들에 hot 데이터 표시 flag를 태그하며, flag가 달린 LSN은 H-table에 기록한다. Hot 데이터들은 빈번하게 접근되며 데이터 갱신이 자주 일어나므로 보통의 임의쓰기와 구분하여 다른 블록에 저장한다. L2 버퍼의 섹터들은 4 섹터씩 업데이트되며 유효한 LSN들만 플래쉬 메모리에 저장된다. 그림 3(b)에서 L2 버퍼의 첫 업데이트 과정을 살펴보면 L2 버퍼의 8섹터가 다 채워졌을 때 유효한 LSN 22, 7, 20이며 이 LSN 들은 비어있던 PBN 3에 매핑된다. PBN 3은 임의 쓰기 블록이며 이 정보가 주 매핑 테이블의 S/R/H field에 R로 기록되며 R-table에는 각 LSN마다 매핑되는 PSN이 기록된다.

3.4 병합 연산 (Merge operation)

메모리가 다 찼을 경우 새로운 데이터를 저장하기 위해서는 병합 연산을 수행한다. 기존의 방식은 여유 블록이 하나 남았을 때 병합 연산을 시작한다. 제안하는 알고리듬은 소거와 copy 횟수를 줄이기 위해서 블록들 중 최대 NIS 값과 여유 블록을 채운 섹터의 개수를 비교한다. 예를 들어, 여유 블록이 하나 남았는데 나머지 블록 중 최대 NIS 값이 3이라면 여유 블록에서 3개의 섹터를 채울 때까지 소거를 delay 시킨 후 블록들 중 최대 NIS 값이 여유 블록을 채운 섹터 수와 같아질 때 병합 연산을 시작한다.

제안된 FTL 알고리듬은 순차 쓰기 블록, 임의 쓰기 블록, 그리고 hot 데이터 블록, 총 3개의 블록을 유지해야 하므로 위의 방식을 확장시킨다. 병합 시 다른 블록으로 copy될 유효한 데이터는 임의 쓰기 형태이므로 순차 쓰기 블록에 copy 할 수가 없기 때문에, 예외적으로 순차쓰기 블록의 경우에는 위와 같은 비교처리를 거치지 않는다. 여유블록이 순차쓰기 블록으로 할당되면 그 블록이 다 채워지지 않았을 지라도 이미 다 채워진 것으로 가정한다. 그림 3(b)에서 순차 쓰기 블록인 PBN 2에 LSN 4,5,6(쓰기 시퀀스 2@ 2# 2$)이 플래쉬 메모리에 기록될 때 블록들 중 최대 NIS 값이 여유 블록을 채운 섹터 수와 같으므로 병합 연산을 시작해야 한다. 최대의 NIS을 갖는 블록이 victim 블록이며, 만약 NIS이 같을 경우 wear-leveling을 고려해 주기 위해서 ECN이 최소인 블록을 victim 블록으로 선택한다. 그림 4는 병합 연산이 수행되는 과정을 보인다. 유효한 LSN을 다른 블록으로 copy 시킨 후 PBN 0를 소거 한다. 이 때 주 매핑 테이블에서 PBN 0의 Logic Block에 대한 정도들 중 ECN만 한 회 증가시키고 나머지들은 소거 한다. 그림 4의 경우는 병합 시 순차 쓰기 블록이 victim 블록으로 선택된 경우지만, 만약 임의쓰기 블록이 victim 블록으로 소거된다면 R-table에서 그 블록에 해당되는 섹터 값들도 소거 시킨다.

그림 5는 쓰기 시퀀스의 2%,2^,2&,2*이 L1 버퍼에 저장된 후 쓰기연산 과정과 병합 연산 과정을 보인다. L1 버퍼의 8 섹터가 다 채워지지 않았으나 그 다음 쓰기 시퀀스가 없을 경우에는 쓰기 과정을 진행한다. 이것은 L2 버퍼의 경우도 마찬가지이다.

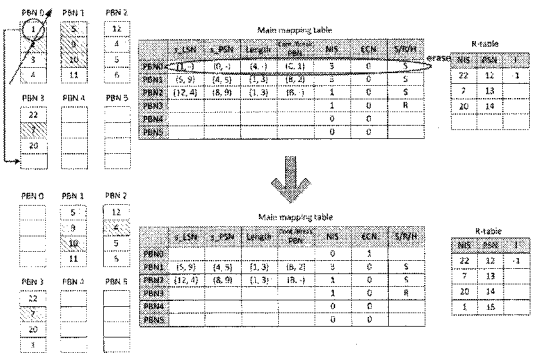


그림 4. 병합 연산.

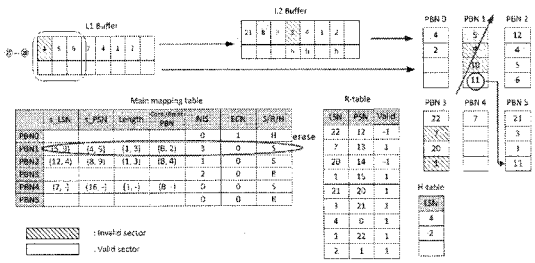


그림 5. 쓰기 시퀀스의 나머지 쓰기 연산과 병합 연산 과정.

Ⅳ. 실험 결과

제안된 알고리듬의 성능을 평가하고 검증하기 위하여 BAST, FAST, 참고문헌 [12]의 결과와 각각 비교하였다. BAST, FAST, 참고문헌 [12]와 제안한 FTL 각각을 시뮬레이터로 구현하여 다양한 트레이스 파일들을 적용하였고, 삼성 K9WBG08U1M large block NAND 플래쉬 메모리를 실험에 사용하였으며, 모델 사양은 표 1과 같다. Windows-XP를 기반으로 한 노

표 1. 실험에 사용한 NAND 플래쉬 메모리 모델 사양

항목	세부사항
NAND Flash model	SLC(Single Level Cell)
Block size	128KB
Page size	2KB
하나의 Block 당 page의 수	64
Read operation access time	25 μsec
Write operation access time	200 μsec
Erase operation access time	2000 μsec

트북 PC에서 MP3 재생기, 동영상 재생기, 웹 브라우저, 문서 편집기의 어플리케이션을 대상으로 실험하여 disk access pattern을 추출하였다.

각각의 어플리케이션에 대한 트레이스 파일의 특징은 표 2와 같다. 실험에 사용한 플래쉬 메모리는 1GB (= 524,288 sectors)의 용량을 사용하였고 쓰기 요구가 발생한 LSN은 전체 LSN의 수에 비해 평균 56.4%이다. 즉, 실제 쓰기가 요구하는 LSN의 수가 전체 LSN 수의 반정도 이므로 동적 할당 기법을 사용하면 효과가 있다. Hot 섹터에 쓰인 LSN의 수는 시간적 지역성을 나타낸다. 전체 쓰기에서 hot 섹터가 평균 67.3% 사용되었으며, Application program #1이 지역성이 가장 높고, Application program #3이 가장 낮은 것으로 나타났다. 이 실험 결과는 disk access pattern이 시간적 지역성이 높으며 이것을 고려해야 FTL의 성능이 나아진다는 것을 의미한다.

표 3은 각각의 어플리케이션에 대한 소거 횟수 비교이며 그림 6은 실험 결과 얻어진 성능을 보인다. 소거 횟수는 알고리듬의 성능을 파악하는데 중요한 파라미터로써 전체 소요 시간과 플래쉬 메모리의 수명을 결정한다. 제안한 FTL을 BAST, FAST, 참고문헌 [12]와 비교하였을 때 평균 72.4%, 61.9%, 10.3%의 소거 횟수가 감소하였다. Application set #1과 같이

표 2. 쓰기 입력 수와 쓰여진 LSN의 수 비교

	# writes	# total LSNs	# LSNs written	# writes at hot sectors
어플리케이션 #1 (MP3 재생기)	683,100	524,288	158,807 (30.3%)	528,036 (77.3%)
어플리케이션 #2 (동영상 재생기)	822,384	524,288	321,389 (61.3%)	587,182 (71.4%)
어플리케이션 #3 (웹 브라우저)	524,288	143,219	364,747 (69.6%)	1,897,680 (56.2%)
어플리케이션 #4 (문서 편집기)	1,262,400	524,288	336,488 (64.2%)	811,723 (64.3%)

* 전체 LSN 중 쓰여진 LSN의 비율을 나타낸다.

표 3. 어플리케이션 별 소거 횟수 비교

	BAST	FAST	참고문헌[12]	제안한 FTL 알고리듬
어플리케이션 #1 (MP3 재생기)	18,381	15,072	4,178	3422 (-81.4%,-77.3%, -18.1%)
어플리케이션 #2 (동영상 재생기)	29,729	19,914	9,314	7793 (-73.8%,-60.9%, -16.3%)
어플리케이션 #3 (웹 브라우저)	220,336	143,219	62,292	59978 (-72.8%,-58.1%, -3.7%)
어플리케이션 #4 (문서 편집기)	68,872	56,591	19,422	18751 (-72.8%,-66.9%, -3.45%)

* 비교 대상과 제안한 FTL의 소거 횟수 증감을 나타낸다.

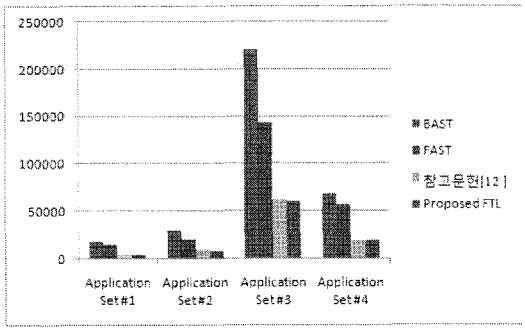


그림 6. 성능 비교

특정 LSN에 대한 쓰기 요구가 중복될 경우 81.4%, 77.3%, 18.1%로 가장 큰 성능 향상을 얻을 수 있었다. Application program #1, #2는 순차 쓰기 요구가 많았으며, application program #3, #4는 임의 쓰기 요구가 많았다. 참고문헌[12]와 제안된 알고리즘은 같은 섹터 매핑 기법을 근간으로 하지만, 제안된 알고리즘은 순차적 지역성과 시간적 지역성을 최대한 활용한다. 임의 쓰기 요구가 많은 경우에는 제안된 알고리듬과 참고문헌 [12]의 성능 차이가 미비하지만 순차 쓰기 요구가 많은 경우 제안된 알고리듬의 성능이 크게 향상된다.

섹터 매핑 기법을 이용하여 제안한 알고리듬은 블록 매핑 기법을 사용했을 때와 비교하여 테이블 크기가 크지만 순차적 지역성의 특성을 이용하고 테이블 업데이트 방식과 동적 할당 방식을 이용하였기 때문에 참고문헌 [12]보다 좋은 성능을 가진다.

V. 결론 및 추후 연구

본 논문에서는 섹터 매핑 기법을 바탕으로 지역성

을 이용하여 순차 쓰기에 유리한 FTL 알고리듬을 제안하였다. 로그 버퍼를 사용하는 기존 FTL 매핑 기법은 섹터 매핑 기법과 블록 매핑 기법을 혼합하여 이용하나 섹터 매핑 기법을 사용하였을 때 보다 소거 횟수 및 소요시간이 증가한다. 제안한 FTL은 섹터 매핑 기법을 기반으로 시간적 지역성과 순차적 지역성을 이용하여 순차쓰기의 효율적인 처리와 임의쓰기의 hot 섹터 추출로 전체 소거 횟수를 감소와 garbage collection의 오버헤드를 줄일 수 있다. 섹터 매핑 기법을 적용 시 매핑 테이블이 커지는 단점을 보완하기 위해서 매핑 테이블 업데이트 방식과 동적 할당 기법을 이용하였다.

실험결과 기존의 알고리듬보다 우수한 성능을 확인하였으며, 추후연구로 순차쓰기에서도 hot 데이터들은 따로 처리하는 방법에 대한 연구가 필요하다. 제안하는 알고리듬의 전체 쓰기 시퀀스에서 hot 데이터들의 비율이 높을지라도 그 데이터가 순차 쓰기 시퀀스에서 속하는 경우 따로 처리해주는 방법이 없다. 순차쓰기 시퀀스에 속한 hot 데이터는 그 안에서 데이터 갱신이 빈번하게 일어나며 많은 수의 무효화 섹터를 발생시킨다. 이 문제점을 해결해 주기 위해서 순차쓰기의 hot 데이터 처리를 위한 연구가 필요하다.

참 고 문 헌

[1] F. Douglis, R. Caceres, M. Kaashoek, K. Li, B. Marsh, and J. Tauber, "Storage Alternatives for Mobile Computers," in Proceedings of the 1st Symposium on Operating Systems Design and Implementation (OSDI), pp.25-37, Nov. 1994.

[2] J. Kim, J. Kim, S. Noh, S. Min, and Y. Cho, "A Space-efficient Flash Translation Layer for Compact Flash Systems,"IEEE Transactions on Consumer Electronics, Vol.48, No.2, pp.366-375, May 2002.

[3] Samsung Electronics, "2G x 8Bit / 4G x 8 Bit / 8G x 8 Bit NAND Flash Memory (K9WBG08U1M) Data Sheets,"2007.

[4] E. Gal and S. Toledo, "Algorithms and Data Structures for Flash Memories," ACM Computing Surveys (CSUR), Vol.37, No.2, pp. 138-163, June 2005.

[5] T. Chung, D. Park, S. Park, D. Lee, S. Lee, and H. Song, "System Software for Flash Memory: A Survey," IFIP Int. Conf. Embedded and Ubiquitous Computing, Lecture Note in Computer Science (LNCS), Springer-Verlag, Vol.4096, pp.394-404, Aug. 2006.

[6] A. Ban, "Flash File System," United States Patent, No.5,404,485, 1995.

[7] A. Ban, "Flash File System Optimized for Page-mode Flash Technologies," United States Patent, no.5,937,425, 1999.

[8] L. Chang and T. Kuo, "An Efficient Management Scheme for Large-Scale Flash Memory storage Systems," ACM Symposium on Applied Computing (SAC), In SAC(2004), pp.862-868, Mar. 2004.

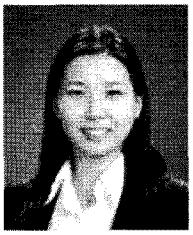
[9] S. Lee, D. Park, T. Chung, W. Choi, D. Lee, S. Park, and H. Song, "A Log Buffer Based Flash Translation Layer Using Fully Associative Sector translation," ACM Transactions on Embedded Computing Systems, Vol.6, No.3, July 2007.

[10] S. Kwon and T. Chung, "An Efficient and Advanced Space-management Technique for Flash Memory using Reallocation blocks," IEEE Transactions on Consumer Electronics, Vol.54, No.2, pp.631-638, May 2008.

[11] S. Lee, D. Shin, Y. Kim, and J. Kim, "LAST: Locality-aware Sector Translation for NAND Flash Memory-based Storage Systems," ACM SIGOPS Operating Systems Review, Vol.42, No.6, pp.36-42, Feb. 2008.

[12] 윤태현, 김광수, 황선영, "섹터 맵핑 기법을 적용한 효율적인 FTL 알고리듬 설계,"한국통신학회 논문지, 제34권 제12호, pp.1329-1543, 2009 년 12월.

홍 수 진 (Soo-Jin Hong) 준회원



2009년 8월 서강대학교 전자공학과 졸업

2009년 9월~현재 서강대학교 전자공학과 석사과정

<관심분야> Flash memory, Embedded System

황 선 영 (Sun-Young Hwang) 정회원

1976년 2월 서울대학교 전자공학과 졸업
1976년 2월 한국과학원 전기 및 전자공학과 공학석사 취득
1986년 10월 미국 Stanford대학교 전자공학 박사학위 취득
1976년~1981년 삼성 반도체(주) 연구원, 팀장
1986년~1989년 Stanford 대학 Center for Integrated Systems 연구소 책임 연구원 및 Fairchild Semiconductor, Palo Alto Research Center 기술자문
1989년~1992년 삼성전자(주) 반도체 기술자문
2002년4월~2004년 3월 서강대학교 정보통신대학원장
1989년 3월~현재 서강대학교 전자공학과 교수
<관심분야> SoC 설계 및 framework 구성, CAD 시스템, Embedded 시스템, DSP 시스템 설계 등