낸드 플래시 메모리상에서 쓰기 패턴 변환을 이용한 효율적인 B-트리 관리

최해기° 박동주
숭실대학교 컴퓨터학부
{hgchoi°, djpark}@ssu.ac.kr

Managing the B-Tree Efficiently using Write Pattern Conversion on NAND Flash Memory

Haegi Choi° Dong-Joo Park
Dept. of Computing, Soongsil University

요 약

플래시 메모리는 하드디스크와 다른 물리적 특성을 가지고 있다. 대표적으로 덮어쓰기가 되지 않고 데이터를 읽고 쓰는 단위와 지우는 단위가 서로 다르다. 이러한 물리적 제약을 소프트웨어적으로 보완해주기 위해서 플래시 메모리를 사용하는 시스템에서는 대부분 Flash Translation Layer (FTL)을 사용한다. 지금까지 FTL 알고리즘의 대부분이 임의 쓰기 패턴보다 순차 쓰기 패턴에 훨씬 더 효율적으로 작용한다. 그러나 B-트리와 같은 자료구조에서는 일반적으로 순차 쓰기 패턴 보다는 임의 쓰기 패턴이 발생된다. 따라서 플래시 메모리상에서 B-트리를 관리할 경우 FTL에 비효율적인 쓰기 패턴을 생성하게 된다. 본 논문에서는 플래시 메모리상에서 B-트리와 같은 자료구조를 효율적으로 저장·관리하기 위한 새로운 방식을 제안한다. 새로운 방식은 B-트리에서 발생되는 임의 쓰기를 플래시 메모리상의 버퍼를 이용하여 FTL에 효율적인 순차 쓰기를 발생시킨다. 실험 결과, 본 논문에서 제안하는 방식은 기존의 방식보다 플래시 메모리에서 발생되는 쓰기 및 블록소거 연산 횟수를 60%이상 감소시킨다.

1. 서 론

플래시 메모리는 크기가 작고 충격에 강하며 하드디스크에 비해 빠른 데이터 접근 속도를 가진다. 또한 소비전력이 적고 무게가 가벼우며 비휘발성의 성질 때문에 PDA, MP3, 핸드폰과 같은 이동기기의 저장장치로 널리 사용되고 있다. 요즘에는 저장크기가 대용량화 되어감에 따라, 개인용 컴퓨터나 노트북에서도 보조기억장치로 활용되고 있다. 이에 따라 대용량의 데이터를 효율적으로 검색하기 위한 B-트리와 같은 인덱스를 플래시 메모리에서 관리하려는 연구들이 이루어지고 있다[1][2].

하드 디스크와는 달리 플래시 메모리는 덮어쓰기(overwrite)가 되지 않는다. 데이터의 읽기·쓰기는 페이지 단위로 이루어지며 소거(erase)는 블록 단위로 이루어진다. 이런 특성 때문에 어떤 페이지에 덮어쓰기가 발생되면, 페이지가 속한 블록(block)을 소거한 후 해당 페이지에 쓰기 연산을 수행해야 한다. 이러한 플래시 메모리의 물리적 제약을 소프트웨어적으로 해결하기 위해 플래시 메모리 저장 시스템에서는 FTL (Flash Translation Layer)을 사용한다[3][4][5]. FTL은 기존 파일 시스템과의 호환성을 위해서 플래시 메모리를 하드 디스크처럼 사용할 수 있게 도와준다. 따라서 플래시 메모리에 덮어쓰기가 발생하더라도 이를 FTL에서 별도의 기법으로 처리하여 사용자에게 플래시 메모리에서 덮어쓰기가 된 것처럼 보여준다.

이러한 플래시 메모리의 물리적 특성을 고려해서 BFTL[1]은 플래시 메모리상에서 B-트리를 생성할 때 덮어쓰기를 최대한 지연시키기 위해서 서로 다른 노드에 새로 삽입된 키 값들을 비어있는 페이지에 함께 저장하였다. 이러한 기법은 덮어쓰기 지연으로 B-트리의 생성 비용을 감소시켜주지만 하나의 노드 안에 있는 키 값들이 서로 다른 페이지에 분산되어 저장되기 때문에 하나의 노드에 접근하기 위해서 여러 개의 페이지를 읽어야 하는 문제점을 가지게 되며, 이로 인해 검색성능이 크게 악화된다.

BFTL의 단점인 검색성능을 향상 시키기 위해서, BOF[2]에서는 하나의 동일한 노드 안의 키 값들을 같은 페이지에 저장시켜 하나의 노드에 접근하기 위해서 하나의 페이지만을 읽는 방법을 제안하였다. 그러나 이 기법은 B-트리를 생성할 때 임의(random) 쓰기 패턴으로 인해 많은 덮어쓰기를 FTL에 발생시켜 플래시 메모리의 전체 연산 비용을 증가시키는 단점을 가지고 있다.

B-트리에 키 값이 삽입·삭제되면, 해당 노드가 수정되고 수정된 노드를 플래시 메모리에 반영하기

* 본 연구는 정보통신부 및 정부통신연구진흥원의 IT신성장동력핵심기술개발사업의 일환으로 수행하였음. [2006-S-040-01, Flash Memory 임베디드 멀티미디어 소프트웨어 기술 개발]

위해, 수정된 노드가 저장되어 있는 페이지에 쓰기 연산이 발생된다. 그런데 임의 키 값들이 삽입(또는 삭제)되면, 노드 수정이 산발적으로 일어나고, 수정된 노드들과 대응되는 페이지들도 산발적으로 생겨나기 때문에 플래시 메모리에 임의 쓰기 패턴이 발생된다. FTL은 임의 쓰기 패턴보다는 순차 쓰기 패턴에 훨씬 더 효율적으로 작용한다. 따라서 B-트리에 임의 키 값들의 삽입(또는 삭제)으로 인해 발생되는 임의 쓰기 패턴은 FTL에 비효율적으로 작용하여 플래시 메모리에서 발생되는 전체 연산 비용을 증가시킨다.

본 논문에서는 플래시 메모리상에서 B-트리를 생성·변경 할 때 플래시 메모리에서 발생되는 연산 비용을 감소시키는 방식을 제안한다. 제안하는 방식은 플래시 메모리의 일부분을 쓰기 버퍼로 사용하여 임의 쓰기 패턴을 순차(sequential) 쓰기 패턴으로 변환하여 FTL에 효율적으로 작용하도록 구성되며, 그 결과 플래시 메모리에서 발생하는 비용을 크게 감소시킨다.

본 논문의 구성은 다음과 같다. 2장에서는 플래시 메모리 특성 및 플래시 메모리상에서 B-트리 관련 연구를 설명한다. 3장에서는 본 논문을 이해하기 위해 알아야 할 예비 지식을 설명하며 4장에서는 제안 동기와 5장에서는 본 논문에서 제안하는 방식을 소개하고 6장에서는 실험 결과를 분석한다. 그리고 7장에서 결론을 맺는다.

2. 플래시 메모리 특성 및 관련 연구

2.1 플래시 메모리 특성

NAND 플래시 메모리의 구성은 그림 1과 같이 다수 개의 블록으로 이루어진다. 블록은 소거(삭제)연산의 최소단위이며 한 블록이 안정적으로 소거될 수 있는 횟수는 십만~백만 번 사이로 제한적이다. 하나의 블록은 32개의 페이지로 구성되며 페이지는 데이터 읽기·쓰기 연산의 최소단위이다. 하나의 페이지 안에는 데이터를 저장하는 512B 크기의 영역, 즉 섹터(sector)가 존재하며 이와 함께 부가정보(ECC, LPN)를 저장하기 위한 16B 크기의 예비 영역(spare area)이 존재한다.

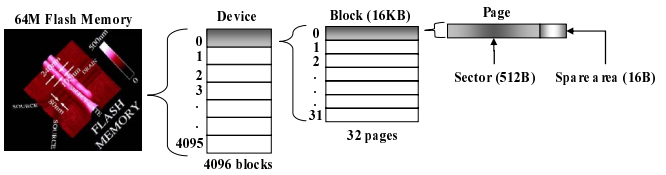


그림 1 플래시 메모리 구성

하드디스크와는 달리 플래시 메모리의 섹터는 덮어쓰기가 되지 않는다. 따라서 이미 데이터가 존재하는 섹터에 새로운 데이터를 저장하기 위해서는 그 섹터를 포함하고 있는 블록을 소거한 뒤 해당되는 섹터에 새로운 데이터를 저장해야 한다. 이러한 물리적 제약을 해결하기 위해 기존의 파일 시스템과 플래시 메모리 사이에는 FTL이 존재한다.

2.2 관련 연구

플래시 메모리의 물리적 특성을 고려하여 B-트리를 저비용으로 저장·관리하기 위한 연구로서 BFTL[1]과 BOF[2]가 존재한다.

BFTL은 덮어쓰기가 되지 않는 플래시 메모리의 특성을 고려하여 서로 다른 노드 안의 새로 업데이트 된 키 값들의 정보를 비어있는 페이지에 같이 저장하며. 이를 위해 유보 버퍼(reservation buffer)와 인덱스 유닛(index unit)을 사용한다. 이렇게 동일한 노드 안에 키 값들이 여러 개의 페이지에 분산되어 저장되기 때문에, BFTL은 각각의 노드 정보를 관리하기 위한 노드 변환 테이블(node translation table)을 사용한다. 이러한 기법은 하나의 노드에 접근하기 위해서 여러 개의 페이지를 읽어야 하므로 B-트리의 검색 성능을 크게 악화시키는 문제점을 가지고 있다. 그리고 노드의 빠른 접근을 위해 노드 변환 테이블을 RAM에 유지하기 때문에 B-트리의 크기가 커졌을 경우 RAM의 비용을 크게 증가시키므로 그 실효성과 현실성이 떨어진다.

BOF는 BFTL에서 많은 비용을 발생시키는 노드 변환 테이블을 없애고 검색 성능을 향상시키기 위해 동일한 노드의 키 값들은 항상 같은 페이지에 저장하는 기법을 사용한다. 즉 하나의 노드에 접근하기 위해서는 한 개의 페이지를 읽으면 된다. 이러한 기법은 비록 B-트리 생성시 BFTL에서 지연시키는 덮어쓰기 연산을 그대로 발생시켜 생성 비용을 증가시키지만 BFTL의 단점으로 여겨진 검색 성능을 크게 향상시키는 결과를 가져온다. 또한 노드 변환 테이블을 사용하지 않기 때문에 B-트리의 크기가 커졌을 경우 필요한 부가적인 비용도 존재하지 않는다. 이러한 이유 때문에 BOF는 BFTL보다 더 유용하고 현실성이 있는 기법이라 할 수 있다. 따라서 이러한 BOF 기법의 기반에서 B-트리 생성 비용을 낮출 수 있는 방안을 연구해야 한다.

3. 예비 지식

3.1 Flash Translation Layer

기존 하드디스크와는 다른 플래시 메모리의 물리적 제약 사항을 소프트웨어적으로 극복하기 위해서 플래시 메모리와 기존 파일 시스템 사이에는 FTL이 존재한다. FTL은 플래시 메모리를 하드 디스크처럼 사용할 수 있게 도와준다. 예를 들어 기존의 파일 시스템에서 이미 데이터가 존재하는 플래시 메모리의 페이지에 새로운

데이터를 쓰려고 하면(즉, 덮어쓰기) FTL이 새로운 데이터를 기존의 데이터가 존재하는 페이지가 아닌 다른 비어있는 페이지에 새로운 데이터를 저장시킴으로써 마치 플래시 메모리에서 덮어쓰기가 이루어진 것처럼 보여준다. 이러한 기능을 위해서 FTL은 사상 테이블(mapping table)을 관리한다.

사상 테이블이란 파일 시스템에서 플래시 메모리의 페이지에 접근(읽기·쓰기 연산)하기 위해 사용하는 논리 페이지 번호(Logical Page Number = LPN)와 플래시 메모리의 실제 물리적 저장 장소인 물리 페이지 번호(Physical Page Number = PPN)를 사상하여 놓은 테이블이다.

3.2 B-트리 노드 수정과 페이지 쓰기와의 관계

일반적으로 B-트리에 키가 삽입(또는 삭제) 되면 키가 삽입(또는 삭제)된 노드 정보가 수정되어야 한다. 플래시 메모리상에서 B-트리를 설계할 때 하나의 B-트리 노드는 플래시 메모리상에서 하나 또는 여러 개의 페이지에 저장될 수 있는데, 본 논문에서는 하나의 노드는 하나의 페이지에 저장된다고 가정한다. 따라서 노드 정보가 수정되면 기존의 노드 정보가 저장된 페이지의 섹터를 수정하는 쓰기 연산(즉, 덮어쓰기)이 FTL에서 발생하게 된다.

4. 제안 동기

4.1 B-트리 쓰기 패턴 분석

B-트리에 임의의 키 값들이 삽입(또는 삭제)되면 키 값들이 삽입(또는 삭제)되는 노드들이 산발적으로 수정된다. 이렇게 산발적으로 수정되는 노드들과 대응되는 플래시 메모리의 페이지들 또한 산발적으로 존재하기 때문에, 해당 페이지들을 수정하기 위해서는 FTL에 임의의 쓰기 패턴을 발생시켜야 한다.

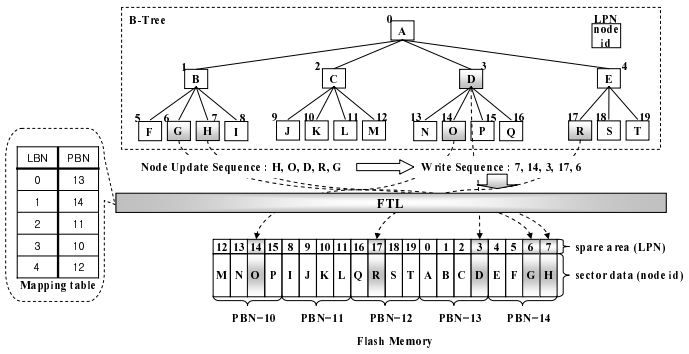


그림 2 B-트리에서 발생하는 임의 쓰기 패턴

그림 2는 플래시 메모리의 하나의 블록이 4개의 페이지로 구성된다고 가정하였을 때, B-트리에 임의의 키 값들이 삽입(또는 삭제)되어 노드 H, O, D, R, G가 수정되었고, 각각의 수정된 노드들과 대응되는 플래시 메모리의 페이지들을 수정하기 위해 논리 페이지 번호 7, 14, 3, 17, 6에 쓰기 연산을 FTL에 발생시키고 있다. 이와 같이 임의의 키 값들이 삽입(또는 삭제)되면 수정된 노드들을 플래시 메모리에 저장하기 위해서 FTL에 임의의 쓰기 패턴이 발생된다.

4.2 FTL에 효율적인 쓰기 패턴 분석

쓰기 패턴에는 크게 순차 쓰기 패턴과 임의 쓰기 패턴으로 나뉘어 진다. 본 절에서는 FTL에 효율적으로 작용하는 쓰기 패턴을 분석하기 위해 디지털 카메라(Kodak DC290)에서 추출한 순차 쓰기 패턴을 임의 쓰기 패턴으로 재구성 하여, 쓰기 패턴에 따른 FTL의 성능을 분석하였다. 그 결과, 그림 3과 같이 순차 쓰기 패턴이 임의 쓰기 패턴보다 FTL에 효율적으로 작용되어, 플래시 메모리에서 발생하는 연산 비용이 감소되는 것을 확인할 수 있었다.

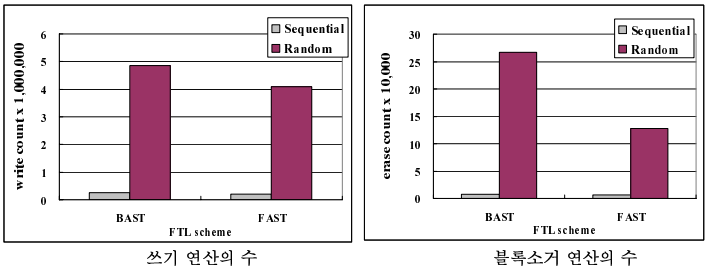


그림 3 쓰기 패턴에 따른 FTL의 성능

B-트리 쓰기 패턴 분석 결과와 FTL에 효율적인 쓰기 패턴 분석 결과를 종합한 결과, B-트리에서 발생되는 임의 쓰기 패턴은 FTL에 비효율적인 쓰기 패턴이어서 플래시 메모리에서 발생되는 연산 비용을 증가시킨다. 따라서 이러한 임의 쓰기 패턴을 FTL에 효율적인 순차 쓰기 패턴으로 변환하면 플래시 메모리에서 발생되는 연산 비용을 크게 감소 시킬 수 있다.

5. 쓰기 버퍼(write buffer)를 이용한 B-트리 생성

본 장에서는 플래시 메모리상에서 B-트리를 생성·변경할 때 FTL에 발생되는 비효율적인 임의 쓰기 패턴을 효율적인 순차 쓰기 패턴으로 변환시키기 위한 방식과 세부적인 알고리듬을 설명한다.

5.1 전체적인 구조

B-트리에서 발생되는 임의 쓰기 패턴을 순차 쓰기 패턴으로 변환시키기 위해서 플래시 메모리의 여러

개의 블록을 쓰기 버퍼로 활용하며 이렇게 활용되는 블록을 쓰기 버퍼 블록(write-page buffer block)이라 한다.

B-트리의 노드가 수정되면, 수정된 노드를 저장하기 위해서 해당되는 페이지에 쓰기 연산이 발생한다. 이렇게 쓰기 연산이 발생된 페이지들을 FTL로 곧 바로 보내면 임의 쓰기 패턴이 발생하기 때문에, 본 논문에서 제안하는 방식에서는 이러한 페이지들을 FTL로 보내지 않고 쓰기 버퍼 관리자로 보낸다. 쓰기 버퍼 관리자는 쓰기 연산이 발생된 각각의 페이지들 마다, 여러 개의 쓰기 버퍼 블록 중에서 쓰기 버퍼 블록 한 개를 선정하여, 선정된 블록 안에 비어있는 페이지에 임시 저장하는데, 이 때 논리 블록 번호가 같은 페이지들은 같은 쓰기 버퍼 블록에 모아진다.

만약 쓰기 버퍼 관리자가 선정한 쓰기 버퍼 블록에 비어있는 페이지가 없으면, 이 블록의 페이지들은 쓰기 패턴 변환기(Write Pattern Converter)에 의해서 순차 쓰기 패턴이 되어 FTL에 발생된 후 소거된다.

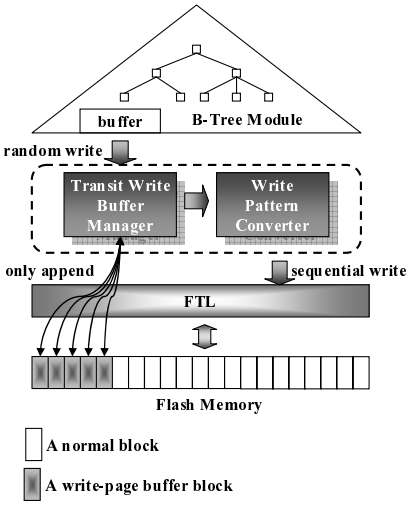


그림 4 전체적인 구조

5.2 쓰기 버퍼 관리자(Transit Write Buffer Manager)

B-트리 모듈에서 수정된 노드를 플래시 메모리에 저장하기 위해 쓰기 연산이 발생된 페이지들은 쓰기 버퍼 관리자에 의해서 쓰기 버퍼 블록에 저장된다. 쓰기 버퍼 관리자는 여러 개의 쓰기 버퍼 블록을 사용하며, 여러 개의 쓰기 버퍼 블록 중에서 쓰기 연산이 발생된 각각의 페이지들마다 저장될 한 개의 쓰기 버퍼 블록을 선정하고 이 블록에 수정된 페이지를 추가 쓰기 한다.

각각의 쓰기 버퍼 블록 안에는 동일한 논리 블록 번호를 가지는 페이지들이 저장된다. 이와 같은 방식으로 페이지들을 쓰기 버퍼 블록에 저장하기 위해서, 쓰기 버퍼 관리자는 쓰기 연산이 발생된 페이지의 논리블록 번호를 계산하여, 해당 논리 블록 번호의 페이지들이 저장되는 쓰기 버퍼 블록을 선정한다. 쓰기 버퍼 관리자에 의해서 페이지가 저장될 쓰기 버퍼 블록이 선정되면 쓰기 버퍼 사상 테이블(Write Buffer Mapping Table - 표 1)을 이용하여 선정된 블록 안에 최초의 비어있는 페이지에 쓰기 연산(즉, 추가 쓰기)을 수행한다. 이와 같은 추가 쓰기(append) 패턴은 블록의 첫 번째 페이지부터 사용하여 순차적으로 다음 페이지를 사용하는 방식으로, 덮어쓰기가 되지 않는 플래시 메모리의 특성에 효율적으로 작용한다.

결국, B-트리 모듈에서 쓰기 연산이 발생된 페이지는 해당 페이지의 논리 블록 번호와 대응하는 쓰기 버퍼 블록에 저장됨으로써, 동일한 논리 블록 번호를 가지는 페이지들은 동일한 쓰기 버퍼 블록 안에 저장된다.

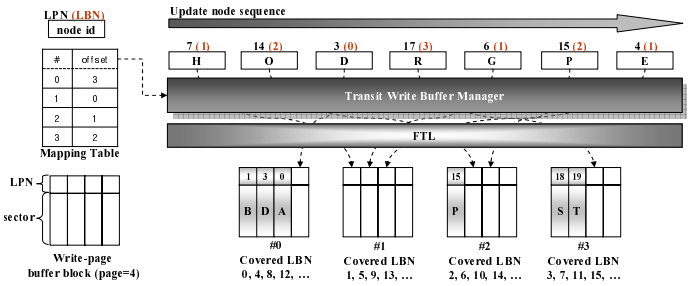


그림 5 쓰기 버퍼 관리자

그림 5는 플래시 메모리의 블록이 4개의 페이지로 구성되며, 4개의 쓰기 버퍼 블록이 존재한다고 가정한다, B-트리 모듈에서 논리 페이지 번호 7, 14, 3, 17, 6, 15, 4에 저장되어 있는 노드 H, O, D, R, G, P, E를 수정하려고 쓰기 연산을 발생시켰을 때, 이 쓰기 연산을 FTL에 곧 바로 보내지 않고 쓰기 버퍼 관리자에 의해서 플래시 메모리의 일부 영역(쓰기 버퍼 블록)에 쓰여지는 과정을 보여준다.

예를 들어, 그림 5에서 논리 페이지 번호 7에 저장되어 있는 노드 H가 쓰기 버퍼 블록에 쓰여지기 위해서 먼저 논리 페이지 번호 7의 논리 블록 번호를 구해야 한다. 논리 블록 번호를 구하는 공식은 (논리 페이지 번호/하나의 블록을 구성하는 페이지 수)의 몫이며, 하나의 블록이 4개의 페이지로 구성된다고 가정하므로, 논리 블록 번호는 1 (7/4의 몫)이다. 그 다음, 논리 블록 번호 1의 페이지들이 저장되는 쓰기 버퍼 블록 번호를 구해야 하는데, 그 공식은 (논리 블록 번호/쓰기 버퍼 블록의 전체 개수)의 나머지 이며, 따라서 논리 블록 번호 1과 대응되는 쓰기 버퍼 블록 번호는 1 (1/4의 나머지)이 된다. 표 1과 같은 쓰기 버퍼 사상 테이블을 이용해서 쓰기 버퍼 블록 1번은 논리 블록 번호 17이라는 것과 블록 안의 최초로 비어있는 페이지가 0번(오프셋)이라는 것을 알 수 있다. 따라서 B-트리 모듈에서 논리 페이지 번호 7에 저장할 수정된 노드 H를 논리 블록 번호 17(쓰기 버퍼 블록)의

0번째 페이지에 임시로 저장하게 된다.

표 1 쓰기 버퍼 블록 사상 테이블

Write-page Mapping Table (existed in RAM)

Write-page Buffer Block Num	LBN	Offset	Page (LPN of node)			
			0	1	2	3
0	2	3	1	3	0	free
1	17	0	free	free	free	free
2	44	1	15	free	free	free
3	82	2	18	19	free	free

5.3 쓰기 패턴 변환기(Write Pattern Converter)

쓰기 버퍼 관리자에 의해서 동일한 논리 블록 번호를 가지는 페이지들은 같은 쓰기 버퍼 블록 안에 저장된다. 쓰기 패턴 변환기는 다 찬 쓰기 버퍼 블록의 최신 페이지들을 순차 쓰기 패턴의 쓰기 연산으로 FTL에 발생시킨다.

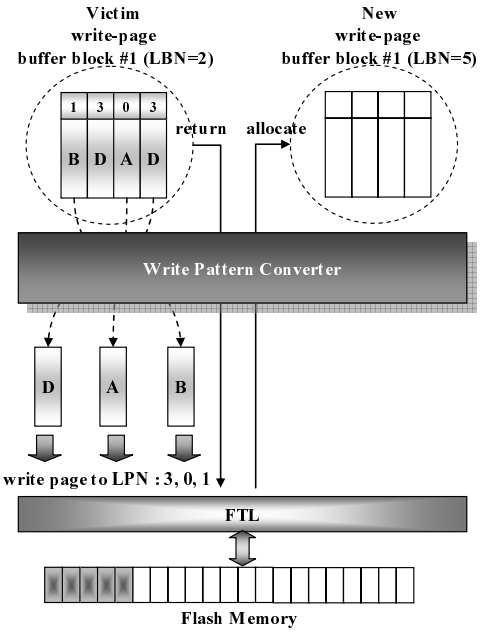


그림 6 쓰기 패턴 변환기

쓰기 패턴 변환기가 순차 쓰기 패턴으로 FTL에 쓰기 연산을 발생시키는 시점은 쓰기 버퍼 관리자가 더 이상 비어 있는 페이지가 없는 쓰기 버퍼 블록(즉, full 쓰기 버퍼 블록)을 사용하려고 할 때이다. 이 때, 쓰기 패턴 변환기는 그림 6과 같이 비어있는 페이지가 없는 쓰기 버퍼 블록을 희생자로 선정하며, 마지막 페이지에서부터 시작하여(backward 방식), 이 블록 안의 최신 페이지들만을 FTL에 쓰기 연산으로 발생시킨다. 예를 들어, 그림 6에서 희생자로 선정된 쓰기 버퍼 블록 안의 두 번째와 네 번째 페이지에 노드 D가 저장되어 있다. 네 번째 페이지에 저장되어 있는 노드 D가 수정된 최신 노드이기 때문에, 두 번째 페이지의 노드 D는 FTL에 쓰기 연산을 발생시키지 않는다.

결국, FTL에 논리 페이지 번호 3, 0, 1과 같은 순서의 쓰기 연산이 발생되며, 이와 같이 희생자로 선정된 쓰기 버퍼 블록 내의 페이지들이 FTL에 쓰기 연산으로 발생되면 동일한 논리 블록 번호를 가지는 페이지들의 쓰기 연산이 발생되기 때문에 임의 쓰기 패턴 보다 순차 쓰기 패턴에 훨씬 더 가까운 쓰기 연산이 FTL에 발생한다.

6. 실험 결과

본 장에서는 논문에서 제안된 쓰기 버퍼를 적용할 때와 적용하지 않을 때의 성능을 분석한다. 플래시 메모리에서 페이지 읽기·쓰기 및 블록 소거 시 소요되는 시간은 표 2와 같다. 페이지 읽기 보다는 페이지 쓰기의 시간이 더 많이 소요되며 여러 페이지를 지워야 하는 블록 소거의 시간이 제일 오래 걸리는 것을 알 수 있다. 이 수치는 Samsung NAND flash chip model K9F1G16U0M 을 실제로 측정한 값[5]이며 이들의 소요 시간 비율은 약 1:3:200 (읽기:쓰기:소거) 이다. 즉 플래시 메모리에서는 쓰기·블록소거 연산이 데이터 처리 시간에 큰 영향을 미친다. 따라서 본 실험에서는 플래시 메모리에서 발생하는 페이지 쓰기와 블록 소거 연산의 횟수를 측정함으로써 쓰기 버퍼의 성능에 대한 우수성을 검증한다.

표 2 플래시 메모리 접근 시간

Operation	Access time
NAND Flash read time (μs/page)	129.72
NAND Flash write time (μs/page)	298.88
NAND Flash erase time (μs/block)	1,998.70

6.1 실험 환경

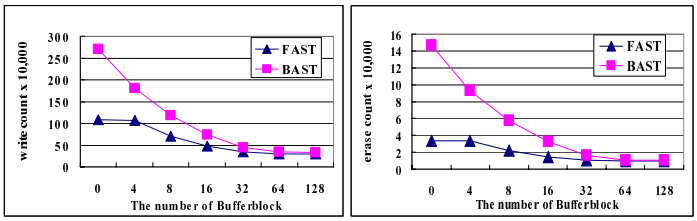
실험 환경은 표 3과 같으며 키 업데이트 횟수와 쓰기 버퍼 블록의 개수를 변화시켜 성능을 측정한다. FTL은 성능이 우수한 BAST, FAST 기법을 모두 사용한다.

표 3 실험 환경

Programming Language	ANSI C
Operating System	Fedora 5
Compiler	gcc version 4.1.0
FTL algorithm	BAST, FAST
The number of Bufferblock	0, 4, 8, 16, 32, 64, 128
The number of key update	50,000 ~ 500,000

6.2 성능 분석

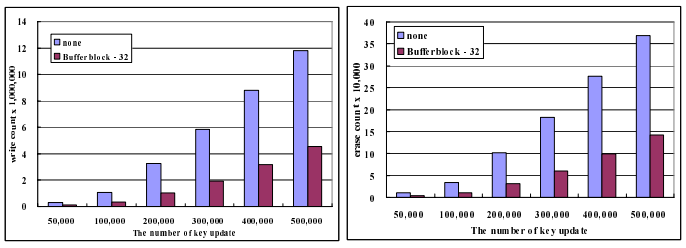
그림 7은 B-트리에 십만 번의 임의 키 값을 업데이트 하였을 경우 플래시 메모리에서 발생되는 쓰기·블록소거 연산 횟수를 나타낸다. X축은 쓰기 버퍼 블록의 개수, Y축은 페이지쓰기 또는 블록소거 연산 횟수를 나타내며 쓰기 버퍼 블록이 0개일 때는 쓰기 버퍼가 사용되지 않을 때를 의미한다. 쓰기 버퍼 블록 개수가 증가할수록 순차 쓰기 패턴이 더 강하게 나타나기 때문에 FTL에 효율적으로 유도되어 플래시 메모리에서 발생되는 비용을 크게 감소시킨다.



(a) 쓰기 연산의 수 (b) 블록소거 연산의 수

그림 7 키 업데이트 횟수 = 100,000

그림 8은 쓰기 버퍼를 사용하지 않을 때와 32개를 사용할 때에 플래시 메모리에서 발생되는 쓰기·블록소거 연산 횟수를 비교한 것이다. FTL은 FAST 기법을 사용하였으며 X축은 업데이트 되는 키의 개수를 의미하는 것으로 오만~오십만 번까지 실험한 것이다. 업데이트 되는 키의 개수에 따라 쓰기 연산 횟수가 60~70%이상 감소하는 것을 볼 수 있다. 이러한 쓰기 연산의 감소는 플래시 메모리에서 발생되는 블록소거 연산에도 영향을 미쳐 블록소거 연산도 60~70%정도 감소함을 알 수 있다. 이러한 블록소거 연산의 감소는 플래시 메모리에서 데이터가 처리되는 시간을 크게 단축시켜주고 또한 플래시 메모리의 각 블록이 안정적으로 소거될 수 있는 횟수가 유한하다는 점을 감안할 때 플래시 메모리의 내구성을 크게 향상 시키는 결과를 가져온다.



(a) 쓰기 연산의 수 (b) 블록소거 연산의 수

그림 8 쓰기 버퍼 블록 = 32 (FAST)

플래시 메모리에서 하나의 블록 크기는 16KB 이다. 따라서 쓰기 버퍼 블록을 32개 사용하더라도 512KB 만을 버퍼 공간으로 사용하게 된다. 이 용량은 4GB의 플래시 메모리를 사용한다고 가정하였을 경우 전체 공간의 0.0125%를 사용하는 것으로 플래시 메모리에서 발생되는 공간 오버헤드가 거의 없다.

7. 결론 및 향후 연구

본 논문에서는 플래시 메모리상에서 B-트리를 생성·변경할 때 발생되는 비용을 낮추기 위해서 쓰기 버퍼를 제안했다. 쓰기 버퍼 블록의 개수가 많을수록 성능은 향상되며, 쓰기 버퍼 블록의 개수가 32개일 때와 쓰기 버퍼를 사용하지 않을 때의 쓰기 및 블록소거 연산 횟수는 업데이트되는 키의 개수와 FTL에 따라 60~85% 정도 감소된다. 또한 플래시 메모리에서 사용하는 버퍼로 인한 공간 오버헤드가 거의 없으며 블록소거 연산의 감소로 데이터 처리 시간을 크게 단축시켜주고 플래시 메모리의 내구성을 향상시킨다. 또한 쓰기 버퍼는 임의 쓰기 패턴을 발생시키는 다른 구조에도 적용될 수 있다.

향후 연구로는 희생자로 선정된 쓰기 버퍼 블록에서 최신 데이터를 모두 FTL에 보내는 방식보다 더 효율적인 알고리듬을 설계하고 쓰기 패턴 외에 FTL에 효율적으로 작용될 수 있는 요소를 연구해서 B-트리의 생성·변경 비용을 개선시킬 수 있을 것이다.

참고문헌

[1] C.-H. Wu, L.-P. Chang, T.-W. Kuo, “An Efficient B-Tree Layer for Flash-Memory Storage Systems,” The 9th International Conference on Real-Time Computing Systems and Applications (RTCSA), 2003.

[2] J. H. Nam, D.-J. Park, "Design and Implementation of the B-Tree on Flash Memory," Korea Information Science Society (KISS), Vol.34, No.2, 2007.

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

[4] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S.-W. Park, and H.-J. Songe. FAST: A log-buffer based ftl scheme with fully associative sector translation. In The 2005 US-Korea Conference on Science Technology, Entrepreneurship, August 2005.

[5] J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee. A Superblock-based Flash Translation Layer for NAND Flash Memory. In the ACM & IEEE Conference on Embedded Software, October 2006.