MFIU : NAND 플래시 메모리상에 B+트리를 위한 효율적인 색인 버퍼 관리 기법

주동수 O* 주영도** 이동호*

한양대학교 컴퓨터 공학과* 강남대학교 컴퓨터미디어공학부

{dsjoo O, dhlee72}@cse.hanyang.ac.kr* ydjoo@kangnam.ac.kr**

MFIU : An Efficient Index Buffer Management Scheme for a B+tree on NAND Flash Memory

Dong-Soo Joo O* Joo, Young Do** Dong-Ho Lee*

Dept. of Computer Science Engineering Han-Yang University*

Division of Computer and Media Engineering Kang-Nam University**

요 약

차세대 저장매체로 떠오르고 있는 플래시 메모리는 가벼운 무게, 작은 부피 그리고 온도 및 충격에 강한 내구성, 적은 전력소모, 빠른 자료 접근성 등의 특징을 가지고 있어 MP3 플레이어, 디지털 카메라, PDA, 핸드폰등과 같은 휴대용 전자기기에 저장장치로 사용되고 있다. 하지만 플래시 메모리가 가지는 하드웨어적 특성 때문에 디스크 기반의 저장장치와는 다른 접근 기법이 필요하다. 특히 B+트리가 구축될 때 레코드의 삽입, 삭제연산 및 노드 분할 연산은 많은 중첩쓰기 연산을 발생하기 때문에 플래시 메모리의 성능을 심각하게 저하시킨다. 본 논문에서는 B+트리의 연산이 수행되는 과정에서 플래시 메모리로 예약버퍼의 색인단위를 반출해야 할 때, 이를 효과적으로 처리 할 수 있는 방법을 제안한다.

1. 서 론

최근 들어 휴대용 전자기기들의 사용이 일반화 되면서 기존의 디스크 기반 저장장치에서는 보장해 주지 못했던 강한 내구성과 저전력형 설계를 가지고 있는 NAND 플래시 메모리가 디지털 카메라, PDA, MP3 플레이어, 핸드폰, PMP등 거의 모든 휴대용 장치들에 주 저장매체로 사용이 되고 있다. 또한 기술의 급격한 발전으로 인하여 저 용량에만 국한되었던 NAND플래시 메모리의 용량이 크게 늘어나 노트북용 저장장치로 하드디스크를 대신할 수 있는 정도까지 활용 분야가 점점 늘어나고 있다.

본 논문은 저장용으로 많이 사용되고 있는 NAND 플래시 메모리를 기반으로 하고 있다. NAND 플래시 메모리는 일정 개수의 블록(block)으로 이루어져 있고, 소 블록(small block) 방식과 대 블록(large block) 방식으로 구분할 수 있다. 소 블록을 경우는 512바이트의 데이터 저장 영역(main area)과 16바이트의 예비 영역(spare area)으로 이루어진 페이지(page)가 32개 모여서 이루어지며 대 블록의 경우는 2048바이트의 데이터 저장 영역과 64바이트의 예비 영역으로 이루어진 페이지 64개가 모여서 이루어진다.

NAND 플래시 메모리는 읽기(read), 쓰기(program), 삭제(erase) 연산을 제공한다. 읽기, 쓰기 연산은 페이지 단위로 이루어지고 삭제 연산은 블록 단위로 이루어지며 연산에 소요되는 시간은 표 1 에서 볼 수 있듯이 비대칭적이다[1]. 또한 NAND 플래시 메모리는 자료를 갱신(update)하기 위해서는 먼저 삭제 연산이 실행 되어야 하는 구조(Erase-before-write Architecture)를 가지고 있다. 때문에 갱신에 필요한 삭제연산을 숨기기 위하여 플래시 변환 계층(Flash Translation Layer)[2, 3]이라는 중간 소프트웨어 계층이 필요하게 된다.

플래시 전환 계층은 논리주소(logical address)를 물리주소(physical address)로 변환해 주는 사상(mapping) 알고리즘을 통하여 갱신 연산시 발생할 수 있는 삭제 연산의 횟수를 줄여줌으로써 플래시 메모리를 효율적으로 사용할 수 있도록 한다. 그러나 대부분의 트리기반(tree-base) 색인 구조는 데이터의 삽입, 삭제 연산 및 노드 분할 연산시 동일한 논리주소에 많은 양의 중첩쓰기 연산을 발생시키기 때문에 플래시 전환 계층만을 사용하는 것은 성능저하를 피할 수 없다.

매체	접근 시간		
	읽기	쓰기	삭제
NAND 플래시	80 ㎲ (2 KB)	200 ㎲ (2 KB)	1.5 ms (128 KB)

표 1. NAND 플래시 메모리의 접근 속도 (Samsung K9WAG08U1A 16Gbit SCL NAND)

일반적으로 많이 사용되고 있는 색인 구조인

B+트리도 레코드의 삽입, 삭제 및 노드 분할 연산시 동일한 논리 주소에 많은 중첩쓰기 연산을 발생시키는 경향이 있다. 따라서 기존의 하드 디스크 상에서 개발된 구조를 그대로 플래시 메모리에 사용하는 것은 비효율적이다. 이러한 문제를 해결하기 위하여 BFTL[4]과 BOF[5]가 제안되었고 이를 더 효과적으로 성능을 향상시킨 IBSF[6]가 제안 되었다.

BFTL은 NAND 플래시 메모리 상에서 B+트리를 효과적으로 구현하기 위하여 예약 버퍼(reservation buffer)와 노드 변환 테이블(node translation table)이란 것을 사용하여 B트리를 플래시 메모리에서 효율적으로 사용할 수 있게 하였다. 예약 버퍼란 B+트리의 노드에 쓰기 혹은 삭제 연산이 일어날 경우 바로 플래시 메모리에 반영하지 않고 호스트 컴퓨터의 메모리에 일정 공간을 할당하여 저장하는 곳이다. 예약버퍼가 다 차게 될 경우 그때서야 플래시 메모리로 모아서 기록하여 쓰기연산의 횟수를 줄여주게 된다. 노드 변환 테이블은 색인단위가 저장된 페이지의 주소를 유지하여 읽기 및 검색 연산을 돕는다. 그러나 BFTL은 노드 변환 테이블을 유지해야 한다는 단점이 존재한다.

이러한 단점을 해결하기 위해서 IBSF는 노드 변환 테이블을 없애고 B+트리에서 연산이 발생하여 각 노드에 기록 되야 할 색인단위를 예약버퍼에 모아 최대한 플래시 메모리로 쓰기 연산이 실행되는 것을 늦춘다. 이때 예약버퍼에 존재하는 노드에 쓰기 혹은 삭제 연산이 발생 하면 그에 해당하는 색인단위를 수정하는 방법을 사용하여 예약버퍼의 활용도를 극대화 한다. 예약버퍼가 다 찬 상태에서 새로운 색인단위가 들어오게 되면 선입선출(FIFO)방식을 사용하여 제일 처음 들어온 색인단위의 노드를 예약버퍼 내에서 모두 찾아 같이 기록하는 방식을 사용하고 있다. 그러나 IBSF기법은 예약버퍼에 존재하는 색인단위들을 플래시 메모리로 반출할 때 단순한 선입선출 방식만을 이용하기 때문에 성능 개선의 여지를 남겨두고 있다.

본 논문에서 제안하는 MFIU는 반출 연산 발생시 예약버퍼의 색인 단위들을 검사하여 가장 많은 색인단위를 가지고 있는 노드를 찾아 모아서 쓰기 연산을 실행하여 성능을 향상시킨다.

본 논문의 구성은 다음과 같다. 2장에서는 기존에 제안되었던 전략들에 관한 내용을 살펴보고 3장에서는 본 논문에서 제안하는 MFIU를 설명한다. 4장에는 기존에 제안되었던 IBSF와 기타 페이지 교체 전략 기법과 본 논문에서 제안하고 있는 전략의 성능을 평가한 후 5장에서 결론과 함께 향후 연구 과제에 대하여 기술한다.

2. 관련연구

2. 1 BFTL

BFTL은 플래시 메모리상에서 효율적으로 B트리를 구축하기 위한 소프트웨어 모듈로 예약버퍼와 노드 변환 테이블로 구성되어 있다. 예약 버퍼는 B+트리의 노드에 쓰기 혹은 삭제 연산이 일어날 경우 바로 플래시 메모리에 반영하지 않고 호스트 컴퓨터의 메모리에 일정 공간을 할당하여 버퍼처럼 사용하는 곳이다. 예약버퍼가 다 차게되면 저장되어 있던 색인 단위들을 플래시 메모리로 모아서 기록하여 쓰기연산의 횟수를 줄여주게 된다. 노드 변환 테이블은 색인 단위가 저장된 페이지의 주소를 메모리에 테이블 형대로 유지하여 읽기 및 검색 연산을 효율적으로 할 수 있게 한다.

B트리가 구축이 되고 노드를 플래시 메모리에 써야 할 경우에 노드의 갱신이 일어날 때마다 바로 플래시 메모리에 갱신하는 것이 아니라 예약버퍼에 모았다가 다 찼을 경우에 플래시 메모리에 쓰는 방식으로 플래시 메모리에서 갱신 연산에 많은 시간을 소모한다는 점을 보강하기 위한 방법이다.

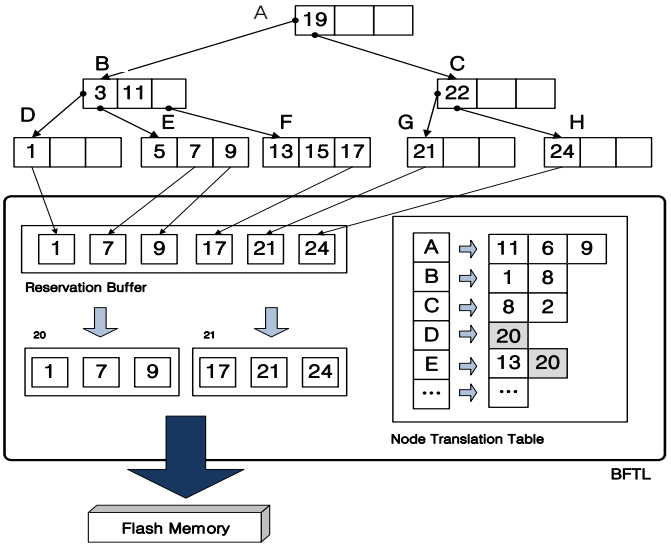

그림 1. BFTL의 구조

BFTL은 색인 단위로 자료를 가공하여 처리하는데 저장될 노드, 값, 연산 등의 정보를 가지고 있다. 그림 1은 1, 7, 9, 17, 21, 24를 입력 받아 플래시 메모리로 반출하는 BFTL의 구조를 예로 나타낸 것이다. 색인 단위로 가공된 자료들이 예약 버퍼에 1차 저장되었다가 꽉 차서 플래시 메모리의 논리 주소 20과 21에 저장되는 모습이다. 저장이 될 때 각 색인 단위들은 노드 변환 테이블에 기록이 된다. 그림 1을 보면 논리주소 20에 기록된 1, 7, 9는 각각 D, E, E의 노드와 노드변환 테이블에 기록이 된다. 따라서 D노드에 저장된 내용을 찾아 가려면 노드변환 테이블의 D노드에 연결된 논리 주소를 찾아 가면된다. 그림에서는 20번지를 가리키고 있으므로 20번지로 가면 D노드에 저장된 값을 찾을 수 있다.

하지만 동일한 노드에 들어가는 색인 단위들이 플래시 메모리 상에 서로 다른 페이지에 저장되기 때문에, 이를 찾아가기 위한 노드 변환 테이블을 계속 유지해주어야 하는 단점이 존재한다.

2. 2 IBSF

IBSF는 BFTL의 단점을 극복하기 위해 플래시 메모리로의 반출 연산시 동일한 노드의 색인단위들을 모아서 써줌으로 노드 변환 테이블을 없애 플래시 메모리상에서 B+트리를 위한 효율적인 색인 버퍼 관리 정책을 제안한 것이다. BTFL에서 제안되었던 예약버퍼의 개념을 가져온 IBSF는 그림 2 에서와 같이 예약 버퍼에 갱신이 일어난 색인단위에 관하여 중복을 제거하여 예약 버퍼를 효율적으로 사용하기 때문에 플래시 메모리로 자료를 기록하는 주기를 최다한 늦추어 플래시 메모리의 쓰기 연산 발생을 낮추어준다.

색인단위는 B+트리의 노드의 삽입, 삭제, 갱신에 대한 정보를 표현하는 단위로 사용이 되며 삽입과 삭제타입의 색인 단위로 구분된다. 색인 단위에서 유지하고 있는 정보에는 데이터 포인터(data pointer), 부모 노드(parent node), 키(key), 왼쪽 포인터, 오른쪽 포인터, 식별자(identifier) 명령어 코드(operation code)로 구성된다. 데이터 포인터는 레코드를 가리키는 포인터이고, 부모 노드는 색인단위 정보를 반영하는 엔트리(entry)의 키이며 왼쪽 포인터와 오른쪽 포인터는 자식노드에 대한 포인터이다. 식별자는 색인단위가 속한 B+트리의 노드를 식별하기 위한 것이고 명령어 코드는 삽입인지 갱신인지를 처리하기 위한 정보이다.

그림 4 에서 볼 수 있듯이 삽입 색인 단위에는 삽입될 데이터와 기타 정보를 삭제될 색인 단위에는 노드의 몇 번째가 삭제 될 것인가를 비트로 표기하였다. 플래그 세트(flag set)의 비트가 1 로 세팅이 되어 있다면 삭제가 될 색인 단위이고 각 노드에서의 위치를 표기하고 있기 때문에 한 노드에서의 여러 삭제 연산을 삭제 인덱스 단위 하나로 표현이 가능하다.

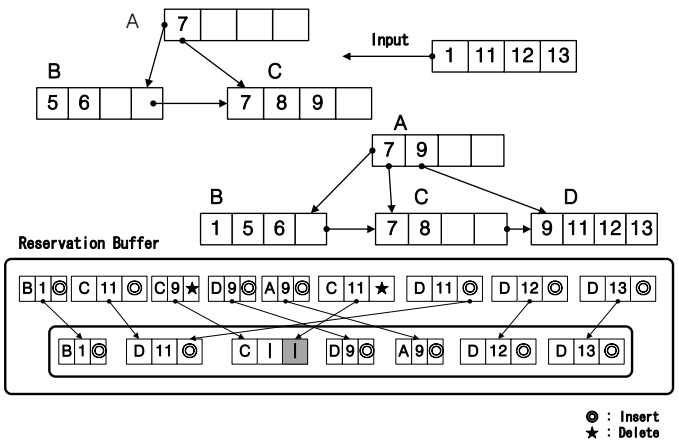

그림 2. IBSF 의 구조

그림 2는 IBSF상에서 자료의 입력예를 보여 주고 있다. 1, 11, 12, 13 이 차례대로 입력이 되면 B+트리가 두번째 트리 처럼 변경이 되고 노드가 분할연산이 발생하게 된다. 이 과정에서 발생하는 자료와 연산 내용은 색인 단위에 기록되어 예약버퍼에 임시 저장된다. 그림 2의 예약 버퍼 내부를 보면 노드 분할 연산의 내용이 각각의 색인 단위로 표현이 되었다. ◎는 삽입을 ★는 삭제 연산을 나타내고 있다. 처음 들어온 값 1은 B노드에 삽입되고 두번째로 들어온 11은 C노도에 삽입된다. 그후 12가 C노드에 삽입되려고 하나 C노드가 다 차게 되여 노드 분할 연산이 일어나게 된다. 9와 11의 색인 단위는 새로 생성된 노드 D로 옮겨 지게 된후 12는 D노드에 삽입된다. 예약 버퍼상에서 C노드에 삽입되기로 했던 11의 색인 단위는 D노드에 삽입되는 것으로 바뀌고 C노드는 11과 12를 지운 표시를 3, 4 번째 비트를 표시함으로 함축 표현한다. 연산이 다 끝난 후 예약 버퍼가 다 차게 되면 선입선출 방식에 의해서 플래시 메모리로 반출연산이 일어난다. 이때 선정된 색인 단위와 같은 노드에 속하는 모든 색인단위는 그림 3에서와 같이 모두 모아서 반출하게 된다.

본 논문에서는 IBSF 의 반출 연산시 예약버퍼에서 사용하는 선입선출 방식을 개선하여 예약버퍼에 존재하는 색인 단위 중 가장 많은 색인 단위가 속하는 노드를 선정하여 그 노드에 속하는 모든 색인 단위들을 모아서 플래시 메모리에 쓰는 전략을 사용한다.

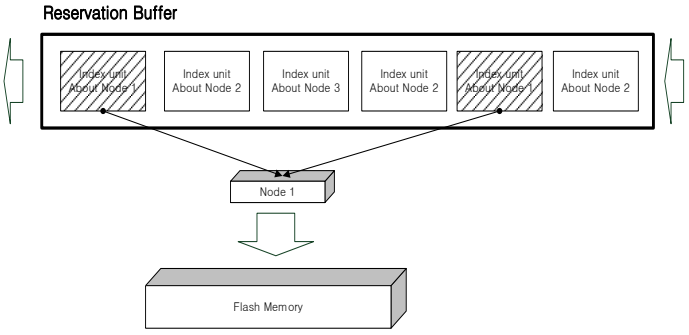

그림 3. IBSF 의 반출 정책

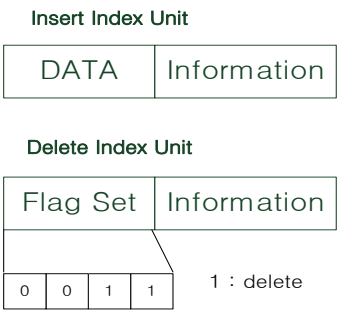

그림 4. 색인단위

3. MFIU 기법

3. 1 MFIU 의 주요 아이디어(Key Idea)

기존의 IBSF에서는 예약 버퍼에 있는 내용들을 플래시 메모리에 쓰고자 할 때 선입선출 방식을 사용하여 예약 버퍼의 가장 앞에 존재하는 인덱스가 속하는 노드에 들어가는 페이지들을 모아서 플래시 메모리에 반출연산을 실행 하였다. 이는 가장 단순한 방식으로 개선의 여지가 있다.

가장 오래된 색인 단위가 예약 버퍼 상에 극히 소수만 존재할 경우에 이는 IBSF전략의 성능을 떨어뜨리는 요인이 될 수 있다. 이를 개선 하기 위해서 본 논문에서는 그림 5 에서와 같이 예약 버퍼에 존재하는 색인 단위 중 가장 많은 색인 단위가 속하는 노드를 선정하여 그 노드에 속하는 색인 단위들을 모아서 플래시 메모리에 쓰는 전략을 사용하였다.

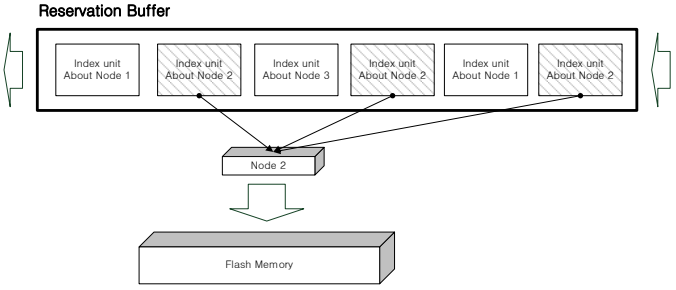

그림 5. MFIU에서의 반출 정책

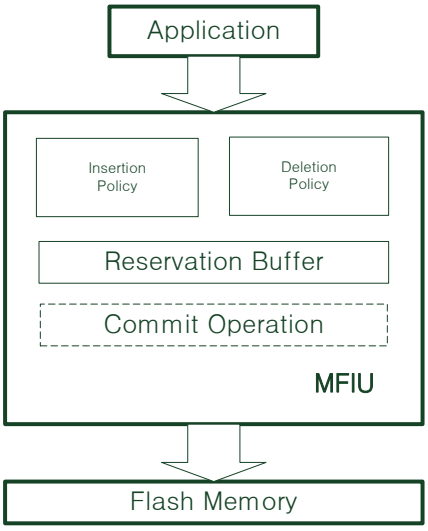

그림 6. 다수 색인 단위 통합 기법(MFIU) 구조

MFIU는 그림 6에서 볼 수 있듯이 응용(application)과 플래시 메모리 사이에 존재하는 소프트웨어 모듈로 구조는 삽입정책(insertion policy), 삭제정책(deletion policy), 예약버퍼, MFIU를 이용한 반출연산(commit operation)이 있다. 구성요소 중 하나인 예약버퍼는 B+트리의 데이터를 플래시 메모리에 저장하기 전 일시적으로 데이터를 유지하기 위한 공간이고 삽입정책, 삭제정책 반출연산과 같은 MFIU의 버퍼관리 기법에 의해 데이터를 처리한다.

3. 2 예약버퍼에서의 페이지 교체 기법

IBSF 에서는 예약버퍼에서 사용하는 페이지 교체 전략으로 선입선출 방식을 사용한다. 선입선출 방식의 페이지 교체 전략은 예약버퍼에 들어온 색인 단위가 다차게 되어서 예약버퍼에 저장되어 있는 색인 단위를 플래시 메모리에 써야 할 때 예약버퍼에 가장 먼저 들어온 색인 단위가 속해있는 노드들과 관련된 모든 색인단위들을 모아서 플래시 메모리에 쓰게 된다. 이러한 방식은 구현하는 방법도 간단하고 색인 단위의 선택도 직관적으로 할 수 있다. 그러나 선입선출 방식의 페이지 교체 전략을 사용한 반출 연산은 불필요한 쓰기 연산을 발생시켜 플래시 메모리 성능을 저하 시킨다. 플래시 메모리의 특성상 성능을 향상시키기 위해서는 가능한 쓰기 연산의 횟수를 줄여 주는 것이 좋다.

반출 연산시 쓰기 성능을 향상시키기 위하여 본 논문에서 제안 하고 있는 방법은 가장 많은 색인 단위를 가지고 있는 노드를 선택하여 반출하는 방법(Most Frequent Index Unit)이다. MFIU 기법의 주요 아이디어는 앞서 기술한 바와 같이 예약 버퍼에 가능한 많은 색인 단위를 가지고 있는 노드를 선택하여 반출하는 것이 추후 플래시 메모리로의 쓰기 연산의 발생을 감소시킬 수 있다는 관찰에 기반한다.

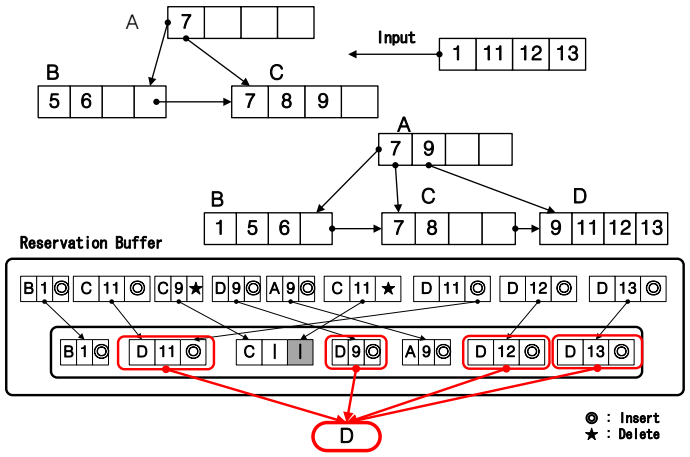

그림 7. MFIU 의 구조

그림 7 은 1, 11, 12, 13 의 값이 B+트리에 입력 될 때의 예를 나타낸 그림이다. 값의 입력 및 노드 분할은 앞서 설명했던 그림 2 의 IBSF 와 동일하게 진행된다. 이 과정에서 예약 버퍼에서 중복이 되는 색인 단위는 하나로 합치고 삭제 연산에 해당하는 색인 단위의 경우에는 플래그 세트에 삭제되는 비트를 1 로 바꾸어 주어 색인 단위의 수를 줄여 주고 있다. 그림 7 에서 볼 수 있듯이 예약 버퍼 내에 있는 C 노드의 색인

단위는 뒤에 두 비트가 1 로 바뀐 모습(음영으로 칠해짐)을 볼 수 있다. 이는 C 노드의 3 번째와 4 번째 값이 지워지는 것을 나타내고 있다.

플래시 메모리로 반출 연산이 일어날 경우 중복을 제거하고 난 후 최종적으로 예약 버퍼에 남는 색인 단위들 중에서 가장 많은 색인 단위가 쓰여질 노드인 D 노드를 찾아 그 노드에 해당하는 색인 단위들을 반출시키는 모습을 그림 7 에서 볼 수 있다.

그림 8 에서 볼 수 있듯이 IBSF 의 경우에는 한 개의 색인 단위를 가지고 있는 B 노드를 선택하여 반출하지만, MFIU 의 경우에는 4 개의 색인 단위를 가지고 있는 D 노드를 선택하여 예약 버퍼 내에 가장 많은 수를 차지하는 색인 단위를 반출 함으로 인해서 예약버퍼의 회전율을 높이고 플래시 메모리에 최적의 쓰기 성능을 보장하여 준다.

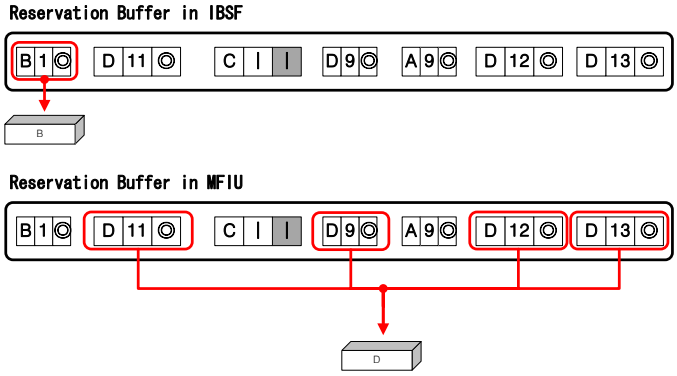

그림 8. IBSF 와 MFIU 비교

4. 실험 및 성능평가

4. 1 성능 평가 환경

성능 평가는 인텔 제온 3.2 GHz 듀얼 프로세스에 2GB 메인 메모리를 가지고 있는 서버급 컴퓨터에서 데비안 리눅스 커널 2.4버전에서 진행되었으며 실험에 사용된 각각의 예제 파일들은 2400개의 레코드로 이루어졌고 B+트리가 유지 할 수 있는 엔트리의 개수는 21로 설정하였다. 실험에 사용한 플래시 변환 계층은 로그 기반의 FAST(fully associative sector translation) [3]이다.

4. 2 예약버퍼의 크기 결정

MFIU 에서 사용된 예약버퍼의 크기는 실험을 통하여 결정하였다. 실험결과는 그림 9 와 같다. 그림에서 x 축은 예약버퍼의 크기를 나타내고 있으며 범위는 10 에서 100 까지다. y 축은 반출이 일어난 횟수를 나타내고 있다. 그래프의 값을 나타내고 있는 것은 입력된 자료의 임의율로 0 은 키값에 따라 정렬하여 입력, 30, 50 ,70, 100 은 각각 30%, 50%, 70% 100%의 임의 입력을 나타내고 있다.

위의 실험에서 버퍼의 크기가 80 이 될 때까지는 일정비율로 감소하나 80 을 넘어 90 으로 증가할 경우 감소폭이 현저하여 버퍼의 크기는 80 으로 설정하였다.

4. 3 반출 횟수 비교

예약 버퍼에서 플래시 메모리로 반출하는 횟수는 플래시 메모리에 쓰기 연산 혹은 삭제 연산을 하는 절대 횟수 이기 때문에 가능한 줄이는 것이 좋다. MFIU의 경우는 예약버퍼에서 매번 최대한으로 색인 단위를 반출 하기 때문에 그림 10 에서 볼 수 있듯이 IBSF에 비하여 같은 예제 파일을 실행 하였을 경우 보다 적은 반출 횟수를 볼 수 있다.

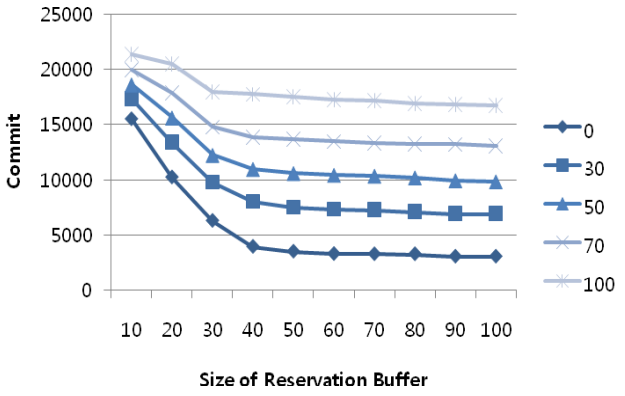

그림 9. 예약버퍼의 크기에 따른 기록 횟수

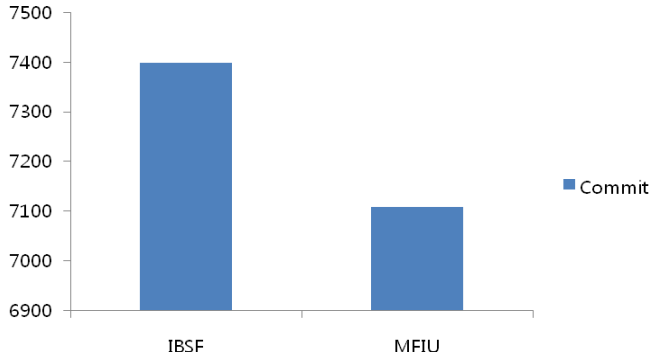

그림 10. IBSF 와 MFIU 의 반출 횟수 비교

4. 4 각 연산 횟수 비교

연산의 비교는 플래시 메모리의 기본 연산인 읽기, 쓰기 그리고 삭제 연산을 비교 대상으로 하였다. 각 연산은 비대칭적인 연산 시간이 걸리고 삭제 연산에 가장 많은 시간이 소모된다. 각 실험에 사용된 파일은 2400 개의 레코드로 이루어진 B+트리용 예제 파일이며 그림 11 의 실험에서는 50% 임의 입력파일이 사용되었으며 그림 12 은 70% 임의, 그림 13 는 100% 임의 파일이 사용 되었다.

표 2 와 그림 11~13 에서 볼 수 있듯이 MFIU 를 사용하였을 경우 IBSF 에 비하여 매 실험마다 3~5 %

정도의 성능 향상을 얻을 수 있었다. 이 결과는 IBSF 에서 사용하는 선입선출 방법 보다 MFIU 의 정책이 더 최적의 성능을 보장한다는 것을 나타낸다.

연산	*IBSF*	*MFIU*	*향상%*
읽기	969997	930133	4.1
쓰기	1667117	1617161	3.0
삭제	55025	52733	4.1

표 2. 50% 임의 파일의 비교 (그림 11)

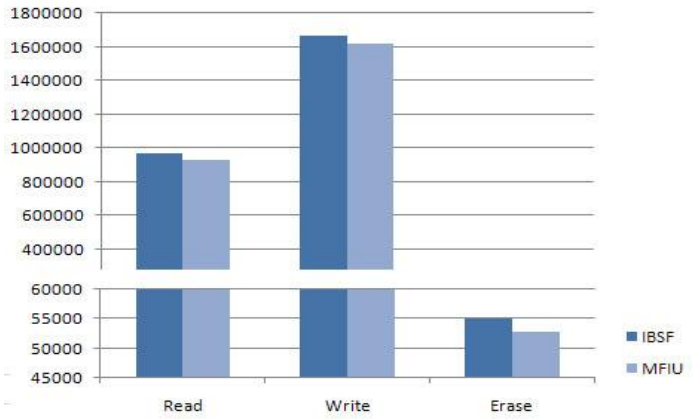

그림 11. IBSF 와 MFIU 의 비교 (50% 임의)

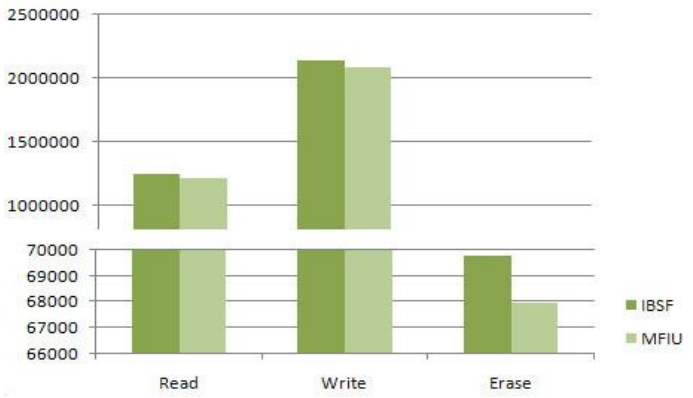

그림 12. IBSF 와 MFIU 비교(70% 임의)

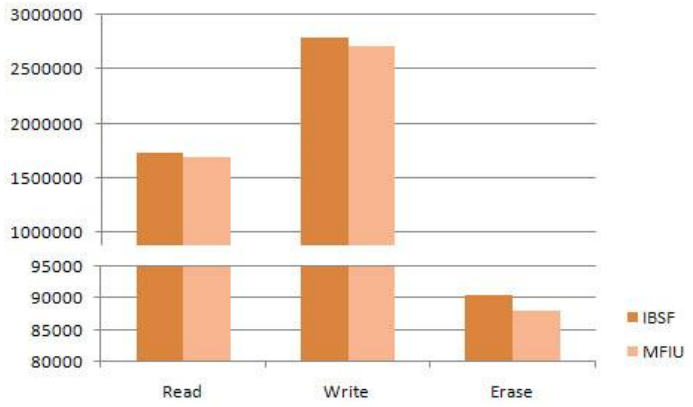

그림 13. IBSF 와 MFIU 비교(100% 임의)

5. 결론

본 논문에서는 효율적인 색인 버퍼 관리 정책인 MFIU를 제안하였다. MFIU는 예약버퍼에 존재하는 색인 단위들을 효율적으로 반출 하는 정책으로 반출 연산 실행시 쓰기 연산을 효율적으로 개선하여 총 수행 시간을 줄 일 수 있었다. 또한 실험을 통하여 MFIU가 IBSF에 비하여 좋은 성능을 보이는 것을 증명하였다.

플래시 메모리는 집중적인 중첩쓰기가 발생할 경우 심각한 성능 저하를 일으킴으로 인덱스와 같이 구축할 경우 이를 보완해 줄 시스템이 필요하다. MFIU는 위의 문제를 보완해 주어 플래시 메모리의 성능을 향상시킬 수 있다.

향후에는 플래시 메모리의 기본 특징인 삭제 연산의 시간을 가능한 줄이는 방향으로 각 연산에 따른 비용기반 연구를 한다면 더 좋은 결과를 볼 수 있을 것이다.

6. Acknowledgement

본 논문은 정통부 및 정보통신연구진흥원의 정보통신 선도기반기술개발사업의 연구결과로 수행되었습니다.

7. 참고문헌

[1] Sang-Won Lee, Bongki Moon, “Design of Flash-Based DBMS: An In-Page Logging Approach,” SIGMOD `07, June 11-14, 2007, Beijing, china.

[2] Intel Corporation, “Understanding the Flash Tanslation Layer(FTL) Specification,” Technical report

[3] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung, Dong-Ho Lee, Sang-Won Park and Ha-Joo Song, “FAST: A Log Buffer based Flash Translation Layer using Fully Associative Sector Translation”, The 2005 US-Korea Conference on Science, Technology, & Entrepreneurship, 2005.

[4] Chin-Hsien Wu, Li-Pin Chang, and Tei-Wei Kuo, “An Efficient B-Tree Layer for Flash-Memory Storage Systems,” Lecture Notes in Computer Science Vol. 2968, pp. 409-230, 2004

[5] 남정현, 박동주, “플래시 메모리 상에서 효율적인 B-트리 설계 및 구현”, 한국정보과학회 05 추계 학술발표논문집(2), pp. 55-57, 2005

[6] 이현섭, 강원식, 이동하, 이동호, “플래시 메모리상에 B+트리를 위한 효율적인 색인 버퍼 관리정책”, 한국정보과학회 학술발표논문집 Vol.33, No.2(C), 2006.