플래시 메모리상에 B+트리를 위한 효율적인 색인 버퍼 관리 정책

이 현 섭[†] · 주 영 도[††] · 이 동 호[†††]

요 약

최근 NAND 플래시 메모리는 충격에 강한 내구력과, 저 전력 소비, 그리고 비휘발성이라는 특징 때문에 MP3 플레이어, 모바일 폰, 노트북과 같은 다양한 이동 컴퓨팅 장비의 저장 장치로 사용되고 있다. 그러나 플래시 메모리의 특수한 하드웨어적 특징 때문에 디스크 기반의 시스템을 플래시 메모리상에 곧바로 적용 하는 것은 여러 단점들을 발생 시킬 수 있다. 특히 B트리가 구축될 때 레코드의 삽입, 삭제연산 및 노드 분할 연산은 많은 중첩쓰기 연산을 발생하기 때문에 플래시 메모리의 성능을 심각하게 저하시킬 것이다. 본 논문에서는 IBSF로 불리는 효율적인 버퍼 관리 기법을 제안한다. 이것은 색인 단위에서 중복된 색인 단위를 제거하여 버퍼가 채워지는 시간을 지연시키기 때문에 B트리를 구축할 때 플래시 메모리에 데이터를 쓰는 횟수를 줄인다. 또한 다양한 실험을 통하여 IBSF 기법이 기존에 제안되었던 BFTL 기법보다 좋은 성능을 보이는 것을 증명한다.

키워드 : 플래시 메모리, B트리, 버퍼 관리 기법, 플래시 전환 계층, 색인 버퍼, 색인 단위

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

Hyun-Seob Lee[†] · Joo, Young Do[††] · Dong-Ho Lee[†††]

ABSTRACT

Recently, NAND flash memory has been used for a storage device in various mobile computing devices such as MP3 players, mobile phones and laptops because of its shock-resistant, low-power consumption, and none-volatile properties. However, due to the very distinct characteristics of flash memory, disk based systems and applications may result in severe performance degradation when directly adopting them on flash memory storage systems. Especially, when a B-tree is constructed, intensive overwrite operations may be caused by record inserting, deleting, and its reorganizing, This could result in severe performance degradation on NAND flash memory. In this paper, we propose an efficient buffer management scheme, called IBSF, which eliminates redundant index units in the index buffer and then delays the time that the index buffer is filled up. Consequently, IBSF significantly reduces the number of write operations to a flash memory when constructing a B-tree. We also show that IBSF yields a better performance on a flash memory by comparing it to the related technique called BFTL through various experiments.

Key Words : Flash Memory, B-tree, Buffer Management Scheme, Flash Translation Layer, Index Buffer, Index Unit

1. 서 론

최근 다양한 모바일 컴퓨팅 기술의 급격한 발전으로 PDA, MP3플레이어, 휴대폰과 같은 내장형 시스템은 더 높은 휴대성을 지닌 효율적인 저장 장치를 요구하게 되었고 NAND 플래시 메모리는 충격에 강한 내구력과 저 전력 소비, 비휘발성이라는 특징 때문에 이러한 내장형 시스템에 적합한 저장장치로 인식되고 있다.

NAND 플래시 메모리 칩은 일정 개수의 블록(block)들로 이루어진 구조를 가지고 있고, 각 블록은 일반적으로 32개의 페이지(page)를 갖는다. 각 페이지는 데이터를 저장하기 위한 512 바이트의 주요 영역과 메타정보를 저장하기 위한 16 바이트의 예비 영역으로 나누어진다. 또한 NAND 플래시 메모리는 읽기, 쓰기, 지우기 연산과 같은 세 가지 기본적인 연산들을 제공하며 읽기 연산은 특정 페이지로부터 데이터를 읽어오는 연산이고, 쓰기 연산은 하나의 페이지에 데이터를 기록하는 연산이고, 지우기 연산은 대상이 되는 블록의 모든 값들을 1로 초기화 시키는 연산이다.

※ 본 연구는 정보통신부 및 정보통신연구진흥원의 IT신성장동력핵심기술개발사업[2006-S-040-01, Flash Memory 기반 임베디드 멀티미디어 소프트웨어 기술 개발]과 과학기술부 및 대구경북과학기술연구원의 연구개발사업의 일환으로 수행하였음.
† 준 회 원 : 한양대학교 컴퓨터공학과 박사과정
†† 정 회 원 : 강남대학교 컴퓨터미디어공학부 교수
††† 정 회 원 : 한양대학교 컴퓨터공학과 조교수
논문접수 : 2007년 6월 5일, 심사완료 : 2007년 8월 14일

NAND 플래시 메모리는 데이터를 갱신할 때 해당 페이지가 지워져 있어야 기록이 가능한 특별한 구조(erase-before-write)를 가지고 있으며 읽기 및 쓰기 연산은 페이지 단위로 처리되는 반면 지우기 연산은 블록단위로 처리되기 때문에 하나의 페이지에 데이터를 기록할 때 해당 페이지가 이미 데이터를 가지고 있다면, 이 페이지를 포함하고 있는 블록을 삭제하여 초기화한 후 데이터를 기록 하는 특징이 있다. 또한 플래시 메모리는 읽기, 쓰기, 지우기 연산의 처리 비용이 비대칭이며 특히 쓰기 와 지우기 연산이 읽기 연산에 높은 비용을 소모하는 특징을 가지고 있다. 이러한 구조상에서 데이터 접근시 읽기와 쓰기 연산만을 사용하는 디스크 기반의 어플리케이션을 곧바로 사용하는 것은 불가능하기 때문에 플래시 전환계층(Flash Translation Layer, FTL)이라고 불리는 소프트웨어가 개발 되었다. 이것은 NAND 플래시 메모리만의 특별한 구조와 특성을 숨기고, 디스크 기반의 저장장치와 같은 환경을 지원해 제공해 주기 위한 드라이버이다. 플래시 전환계층의 주요 역할은 응용 프로그램이 데이터를 유지하고 있는 페이지로 데이터를 기록하려고 할 때 비어있는 페이지에 데이터를 기록하도록 주소를 재 사상 시켜주는 것이다.

플래시 전환계층을 통해 중첩 연산이 발생했을 때 논리적인 주소를 물리적인 주소로 사상시킴으로써 자기 디스크와 같은 환경을 제공해 주어도 많은 중첩쓰기 연산이 발생했을 경우 성능이 저하되는 것을 피할 수 없다. 특히 특정 주소로 반복적인 중첩연산을 발생하는 경향이 있는 트리 기반의 색인 구조는 구축될 때 많은 무효화 페이지를 생성하기 때문에 성능저하를 발생시킬 수 있다.

이러한 문제를 해결하기 위해 BFTL 기법이 제안되었다. 이것은 NAND 플래시 메모리 상에 B트리를 효율적으로 구축하기 위한 소프트웨어 모듈로써 B트리에 데이터가 삽입, 삭제되어 발생하는 노드분할이나 재분배시 노드에 대한 집중적인 중첩쓰기 연산을 효율적으로 처리하기 위한 구조를 가지고 있다. BFTL의 구조는 예약 버퍼(Reservation Buffer)와 노드 변환 테이블(Node Translation Table)로 구성된다. 예약 버퍼는 일정 개수의 데이터를 유지하기 위해서만 사용되며 BFTL은 B트리 구축시 변경된 정보를 곧바로 플래시 메모리에 저장하지 않고 색인 단위로 생성하여 일시적으로 예약버퍼에 유지한다. 그리고 버퍼가 채워지면 유지하고 있던 색인 단위들을 몇 개의 페이지에 모아서 기록한다. 따라서 하나의 페이지에 관련된 색인 단위는 여러 페이지에 분할되어 저장 될 수 있으며 여러 페이지로 분산된 색인 단위들을 효율적으로 관리하기 위해 노드 변환 테이블을 사용한다. 노드 변환 테이블은 각 노드마다 각 노드에 관련된 색인 단위들이 어느 페이지에 저장되어 있는지 주소 정보를 유지하고 있는 테이블로써 B트리에 데이터를 쓰거나 지울 때에는 노드 변환 테이블의 정보를 이용하여 노드에 관련된 정보를 유지하고 있는 페이지에 접근할 수 있다. 그러나 이러한 기법은 몇 가지 문제를 가지고 있다. 첫째, BFTL은 특정 노드를 논리적으로 완전하게 구축하기 위해 여러 페이지를 접근해야 하기 때문에 많은 읽기 연산을 발생시킨다. 둘째, 노드변환 테이블을 관리 하는데 많은 오버헤드가 든다. 셋째, BFTL은 예약버퍼에 데이터를 저장할 때 아무런 관리기법을 사용하지 않기 때문에 노드분할이나 재분배시 중복된 데이터가 존재할 수 있다. 이러한 문제들은 B트리를 접근하거나 버퍼에서 데이터를 플래시 메모리로 데이터를 기록할 때 많은 읽기 및 쓰기 연산을 발생 시킬 수 있기 때문에 성능이 저하되는 원인이 될 수 있다.

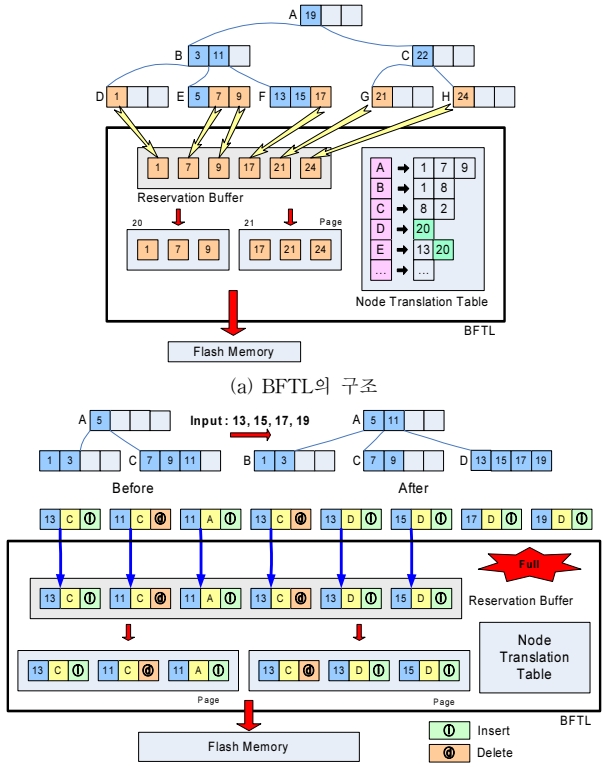
본 논문에서는 BFTL의 단점을 극복하기 위하여 IBSF라는 새로운 버퍼 관리정책을 제안한다. IBSF는 색인 단위를 버퍼에 저장할 때 버퍼에서 중복된 데이터를 제거하기 때문에 커밋 주기를 늦추어 쓰기 연산의 수를 줄일 수 있다. 또한 하나의 노드에 관련된 색인 단위들은 하나의 페이지에 저장하기 때문에 특정 노드를 논리적으로 구축할 때 한 번의 읽기 연산만으로 구축이 가능하다. 따라서 읽기 연산의 수를 BFTL 기법보다 줄일 수 있고, 관리하는데 오버헤드가 큰 노드 변환 테이블을 제거할 수 있기 때문에 BFTL 기법보다 저비용으로 NAND 플래시 메모리에 B트리를 구축할 수 있다.

본 논문의 구성은 다음과 같다. 2절에서는 플래시 메모리 상에 B트리를 구축하기 위한 BFTL에 관하여 간략하게 검토하고, 그것이 가지고 있는 문제점에 대하여 기술한다. 3절에서는 본 논문에서 제안하는 IBSF의 핵심 아이디어와 버퍼 관리 정책에 관하여 자세하게 설명한다. 4절에서는 실험 및 실험 결과를 보이고, 마지막으로 5장에서 결론을 맺는다.

2. 관련 연구

2.1 BFTL의 개요

BFTL은 전통적인 플래시 전환계층 위에 삽입 가능한 소프트웨어 모듈로써 플래시 메모리 상에 B트리를 효율적으로 구축하기 위해 제안되었다. (그림 1)의 (a)는 BFTL을 사용하는 시스템의 구조를 보여주고 있다. 그림과 같이 BFTL은 예약버퍼, 노드 변환 테이블로 이루어져 있다. 어플리케이션에서 B트리의 노드에 데이터를 삽입하거나 삭제할 때 이 정보는 색인 단위로 표현되고 BFTL은 발생한 색인 단위를 바로 플래시 메모리상에 저장하지 않고 예약버퍼에 일시적으로 유지 한다. 예약 버퍼는 일정한 개수의 색인 단위를 유지할 수 있기 때문에 예약버퍼가 채워지면 버퍼가 유지하고 있던 색인 단위들을 플래시 메모리로 커밋 해야 한다. 색인 단위들은 하나의 페이지의 크기보다 상대적으로 작아서 B트리 노드의 많은 색인 단위들은 몇 개의 페이지에 모아서 저장하고, 이 방법은 플래시 메모리에서 쓰기 연산의 횟수를 줄인다. 그러나 하나의 B트리 노드에 관련된 색인 단위들이 여러 물리적인 페이지에 나뉘어 저장 될 수 있기 때문에 임의의 B트리 노드를 구축하기 위해서는 그 노드에 관련된 색인 단위들을 저장한 페이지들을 모두 읽어야 한다. 노드 변환 테이블은 B트리의 노드에 관련된 색인 단위들이 저장된 페이지의 주소를 관리하기 때문에 노드를 논리적으로 구축하는 작업을 효율적으로 만든다. (그림 1)의 (a)는 B트리 구축시 BFTL에서 처리하는 과정을 보여주고 있



(a) BFTL의 구조

(b) BFTL의 단점

(그림 1) BFTL을 사용하는 시스템의 구조

다. (그림 1)의 (a) 예제와 같이 키 값이 1, 7, 9, 17, 21, 24인 데이터가 B트리에 삽입이 되면 BFTL은 이것을 반영하기 위해 6개의 색인 단위를 생성하고 생성한 색인 단위들을 예약버퍼에 저장한다. 본 논문의 예제에서 예약버퍼가 유지할 수 있는 개수는 6개이고 하나의 페이지에는 3개의 색인 단위를 저장할 수 있다고 가정하였다. 예약버퍼가 다 채워지면 BFTL은 버퍼가 유지하고 있는 색인 단위들을 모아서 플래시 메모리상의 몇 개의 페이지에 저장하는 커밋 연산을 수행한다. (그림 1)의 (a) 예제에서 BFTL은 6개의 색인 단위들을 2개의 페이지 20번 21번 페이지에 모아서 저장을 하고 A, E, F, G, H노드에 관련된 색인 단위들이 플래시 메모리의 어느 주소에 저장되었는지 노드 변환 테이블을 갱신한다. 노드 변환 테이블은 새로운 주소정보를 링크 형식으로 유지하기 때문에 기존의 정보에는 영향을 미치지 않는다. 예를 들어 E노드에 관련된 색인 단위는 13번 페이지에만 저장 되어 있었다. 그러나 방금 전 커밋 연산으로 인하여 20번 페이지에도 키 값이 7, 9인 색인 단위가 저장되었기 때문에 20번 페이지에 데이터를 저장하고 있다는 것을 링크 형식으로 추가한다.

만약 (그림 1)의 (a)에서 BFTL을 사용하지 않았다면 1, 7, 9, 17, 21, 26의 키 값을 가지고 있는 데이터가 B트리에 삽입되었을 때 이것을 처리하기 위해 모두 6번의 쓰기 연산이 필요했을 것이다. 그러나 BFTL은 색인 단위를 모아서 2개의 페이지에 저장했기 때문에 2번의 쓰기 연산만으로 B트리를 구축 할 수 있다.

2.2 BFTL의 단점

BFTL은 다음과 같은 몇 가지 단점을 가지고 있다. 첫째, 하나의 노드에 관련된 색인 단위들은 플래시 메모리상에 여러 페이지에 나뉘어 저장될 수 있기 때문에 하나의 노드를 논리적으로 재구성하기 위해 관련된 페이지들을 모두 읽어야 한다. 이것은 B트리의 노드를 접근할 때 많은 읽기 연산을 발생시키기 때문에 성능을 저하 시킬 수 있다. 예를 들어 (그림 1)의 (a)에서 노드 E를 재구성하기 위해 BFTL은 페이지 13과 20을 읽어야 한다.

두 번째, 노드 변환 테이블과 그것이 유지하고 있는 리스트는 RAM에 유지되고, 빠른 속도로 증가한다. BFTL의 저자는 이 리스트들의 길이 너무 길어지면 많은 자원을 소비하기 때문에 일정 길이를 넘어서면 압축하는 방법을 제안하였으나 플래시 메모리에서 압축할 대상이 되는 리스트의 모든 색인 단위를 읽은 후 이것을 재구성하여 플래시 메모리에 재 저장 하는 것은 추가적인 읽기 및 쓰기 오버헤드이다.

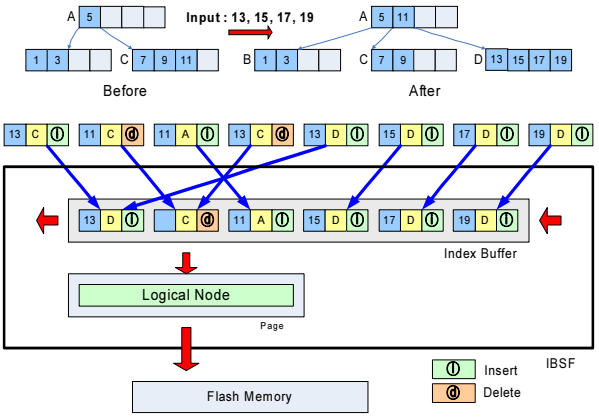
마지막, BFTL은 B트리를 구축할 때 버퍼 관리 정책을 사용하지 않고 색인 단위를 발생한 순서대로 버퍼에 저장하기 때문에 노드 분할이나 재분배 시 버퍼에 중복된 데이터가 유지될 수 있다. (그림 1)의 (b)는 BFTL이 노드분할 연산을 처리하는 과정을 보여주고 있다. 그림에서 색인 단위는 [키 값, 노드 식별자, 명령코드]로 표현된다. 명령코드 ⓘ와 ⓓ는 각각 색인 단위가 삽입을 위한 것인지 삭제를 위한 것인지 나타내는 타입명이다. 예를 들어 [13, C, ⓘ]는 C노드에 키 값이 13인 데이터를 삽입하라는 것을 의미한다. (그림 1)의 (b)의 예약버퍼는 키 값이 13인 색인 단위를 중복해서 유지하고 있다. 만약 중복된 데이터를 버퍼가 유지한다면 가장 최근에 생성된 데이터만 유효한 값을 지니게 되기 때문에 중복된 데이터들은 버퍼의 자원을 낭비하여 커밋 하는 주기를 빠르게 만들고 추가적인 쓰기 연산을 발생할 수 있다.

3. IBSF의 설계 및 적용

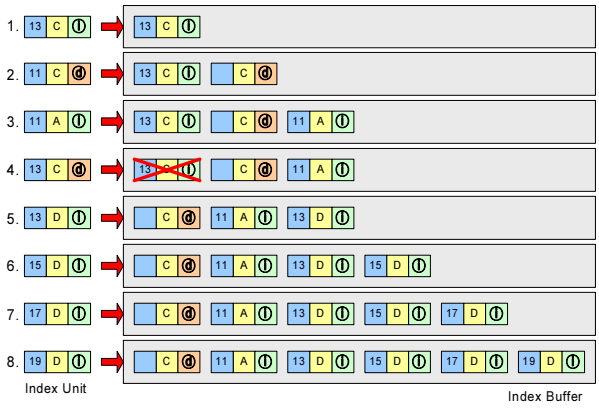
3.1 핵심 아이디어

IBSF의 핵심 아이디어는 다음과 같이 두 가지로 정리할 수 있다. 첫 번째 아이디어는 하나의 B트리 노드에 관련된 색인 단위는 하나의 페이지에 저장한다. 이 방법은 하나의 페이지를 읽는 것으로 하나의 노드를 구축할 수 있기 때문에 추가적인 오버헤드를 발생 하는 노드 변환 테이블을 제거할 수 있다. 두 번째 아이디어는 색인 버퍼에서 중복된 색인 단위를 제거하는 것이다. 이 방법은 버퍼가 채워지는 시간을 늦추기 때문에 플래시 메모리상에 쓰기 연산의 횟수를 줄일 수 있다. 이 두 가지 아이디어를 통해 IBSF는 플래시 메모리상에 효과적으로 B트리를 구축할 수 있다.

(그림 2)의 (a)는 IBSF의 구조를 보여주고 있다. IBSF는 색인 버퍼, 삽입 정책, 삭제 정책, 커밋 정책으로 구성되며 B트리에 데이터의 삽입이나 삭제 변경 시 노드의 변경을 반영하는 색인 단위들을 생성하고 생성된 색인 단위들을 IBSF의 삽입, 삭제 정책에 따라 색인 버퍼에 일시적으로 유



(a) IBSF의 구조



(b) IBSF의 색인 단위를 관리하는 과정

(그림 2) IBSF의 구조

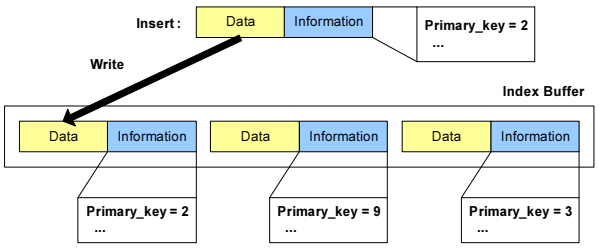
지한다. 버퍼는 정해진 개수의 색인 단위 만을 유지할 수 있기 때문에 버퍼가 다 채워지면 색인 단위들을 커밋 정책에 따라 플래시 메모리로 기록한다. 커밋 연산을 수행할 때 IBSF는 한 노드에 관련된 색인 단위들은 하나의 페이지에 모아서 저장하기 때문에 추가적인 오버헤드를 발생하는 노드 변환 테이블을 사용하지 않는다. 이것은 노드 변환 테이블에의해 발생하는 오버헤드 제거뿐만 아니라 B트리 접근 시 하나의 페이지를 읽는 것만으로 하나의 노드를 구축 할 수 있기 때문에 한 개의 노드를 구축하기 위해 여러 페이지를 읽어야 하는 BFTL에 비해 읽기 성능을 향상시킬 수 있다. 예를 들어 (그림 1)의 (b)에서 BFTL은 노드 A를 구축하기 위해 1, 7, 9번 페이지를 읽어야 한다. 그러나 IBSF는 A노드에 관련된 색인 단위들을 한 개의 페이지에만 저장하기 때문에 A노드를 구축하기 위해 하나의 페이지만 읽으면 된다. 또한 IBSF는 색인 버퍼에서 중복된 색인 단위를 제거하기 때문에 커밋이 되는 시간을 지연시킬 수 있다. (그림 1)에 (b)와 (그림 2)의 (a)는 BFTL과 IBSF를 사용하는 각각의 시스템에서 13, 15, 17, 19의 키 값을 갖고 있는 데이터가 B트리에 삽입되는 과정을 보여주고 있다. (그림 1)의 (b) 예제에서 키 값이 13인 데이터가 삽입될 때 아무런 일도 발생하지 않고 C노드에 삽입할 수 있었다. 그러나 키 값이 15인 데이터가 삽입될 때 B트리의 노드가 가득 찼기 때문에 노드 분할이 발생하고 이것을 처리하기 위해 5개의 색인 단위를 생성한다. 결과적으로 (그림 1)의 (b) 예제에서 입력된 데이터를 B트리에 모두 삽입하기 위해서 8개의 색인 단위를 생성 하였다. 이렇게 발생한 색인 단위들을 만약 BFTL을 사용하여 처리 한다면 특별한 버퍼 관리 정책을 사용하지 않고 생성된 순서대로 버퍼에 색인 버퍼를 삽입하기 때문에 6개 이상의 색인 단위가 입력되었을 때 커밋을 한다. (그림 1)에 (b)의 예제에서 발생한 색인 단위중 [13, C, ⓘ], [13, C, Ⓓ], [13, D, ⓘ]는 키 값이 13인 데이터에 대한 중복된 색인 단위 이다. 이 색인 단위들 중 의미가 있는 색인 단위는 [13, D, ⓘ]이고, [13, C, ⓘ], [13, C, Ⓓ]는 오래된 정보를 표현하기 때문에 중복된 색인 단위를 유지하기 위해 버퍼를 사용하는 것은 불필요한 자원 낭비이다. 이러한 문제를 개선하기 위해 IBSF는 (그림 2)의 (a) 예제와 같이 버퍼에 색인 단위를 삽입할 때 중복된 색인 단위를 제거하기 때문에 8개의 색인 단위가 버퍼에 삽입 되어도 커밋 연산 없이 모든 색인 단위들을 처리할 수 있다.

(그림 2)의 (b)는 (그림 2)에 (a)에서 데이터의 삽입으로 인해 노드 분할이 일어났을 때 IBSF가 발생된 색인 단위를 버퍼에 삽입하기 전에 처리하는 방법과 과정을 단계적으로 보여주고 있다. (그림 2)의 (b) 1, 2, 3번 단계에서 [13, C, ⓘ], [11, C, ⓓ], [11, A, ⓘ]는 중복된 데이터가 없기 때문에 아무런 처리 없이 순서대로 버퍼에 삽입되지만 4번 단계에서 [13, C, ⓓ]가 삽입 될 때 버퍼는 이미 [13, C, ⓘ]를 가지고 있고 이것은 키 값이 13인 데이터에 대한 중복된 색인 단위 이다. 중복된 데이터가 존재할 경우 오래된 색인 단위의 정보는 버퍼에 유지할 필요가 없기 때문에 IBSF는 불필요한 데이터인 [13, C, ⓘ]를 버퍼에서 제거한 다음 [13, C, ⓓ]를 버퍼에 저장한다. 이때 [13, C, ⓓ]는 C노드에 대한 삭제 정보를 담고 있는 색인 단위를 의미한다.

IBSF는 삭제정보를 담고 있는 색인 단위를 삽입정보를 담고 있는 색인 단위와 다르게 처리한다. 삭제 정보는 B트리의 색인 데이터가 필요하지 않고 특정 노드내의 어느 데이터를 삭제해야 하는지 위치 정보만을 필요로 하기 때문에 삭제 타입의 정보를 담고 있는 색인 단위는 0과 1로 위치 정보를 표시할 수 있는 일련의 비트열 형태로 처리할 수 있으며, IBSF는 이 방법을 통하여 삭제할 여러 데이터의 위치 정보를 하나의 색인 단위에 저장하기 때문에 하나의 삭제타입 색인 단위는 하나의 노드에 관련된 삭제 정보를 유지할 수 있다. (그림 2)의 (b) 5번째 단계 예제에서 [13, C, ⓓ]가 삽입될 때 버퍼에는 이미 C노드의 삭제 정보를 유지하고 있는 색인 단위 [, C, ⓓ]가 존재하기 때문에 [13, C, ⓓ]는 [, C, ⓓ]에 비트 정보를 통하여 추가된다. 그 다음 [13, D, ⓘ], [15, D, ⓘ], [17, D, ⓘ], [19, D, ⓘ]는 버퍼에 중복된 색인 단위가 없기 때문에 아무런 처리 없이 버퍼에 삽입된다. 더 자세한 삽입, 삭제 및 커밋 정책은 다음 절에서 설명한다.

3.2 삽입 정책

B트리 노드에 데이터의 삽입이 발생했을 때 IBSF는 새롭게 생성된 삽입타입의 색인 단위를 다음과 같은 일련의




```
1.  insert(key, data) {
2.      삽입을 반영하는 색인 단위 생성
3.      색인 버퍼에서 중복된 색인 단위 검색
4.      if(중복된 색인 단위가 존재) {
5.          중복된 색인 단위를 갱신
6.      } else {
7.          새로운 색인 단위를 버퍼에 삽입
8.      }
9.  }
```

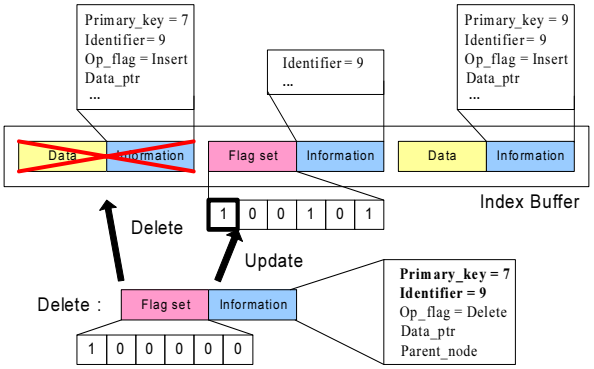
(그림 3) IBSF의 삽입정책

처리 과정을 거쳐 색인 버퍼에 저장한다. 첫째, B트리 노드에 데이터가 삽입되었을 때 이것을 반영하기 위한 색인 단위를 생성하고 생성한다. 그 다음 IBSF는 색인 버퍼에서 새로운 색인 단위의 키 값과 동일한 키 값을 가지고 있는 색인 단위를 검색한다. 만약 동일한 키 값을 가지고 있는 색인 단위가 버퍼에 존재한다면 이것은 새로운 색인 단위와 중복된 색인 단위 이다. 따라서 IBSF는 색인 버퍼에 유지하고 있던 색인 단위를 제거하고 새로운 색인 단위를 버퍼에 삽입한다. 둘째, 만약 색인 버퍼에 동일한 키 값을 가지고 있는 색인 단위가 없다면 새로운 색인 단위를 저장하기 위해 버퍼에 자원을 할당한다. 셋째, 만약 버퍼가 이미 모든 자원을 할당하였기 때문에 더 이상 새로운 색인 단위를 저장할 수 있는 자원이 없다면 커밋 정책에 따라 색인 단위들을 플래시 메모리로 기록한다. (그림 3)은 IBSF에서 삽입 연산이 처리되는 과정을 보여주고 있다. (그림 3)의 예제와 같이 키 값이 2인 색인 단위가 B트리 노드의 데이터 삽입을 반영하기 위해 생성되었다면 IBSF는 발생한 색인 단위를 처리하기 위해 색인 버퍼에서 중복된 색인 단위가 존재하는지 색인 단위가 유지하고 있는 키 값을 이용하여 조사를 한다. 예제에서 색인 버퍼는 3개의 색인 버퍼를 유지하고 있다고 가정하였고 첫 번째 색인 단위의 키 값은 2, 두 번째 색인 단위의 키 값은 9, 세 번째 색인 단위의 키 값은 3이라고 가정하였다. IBSF는 색인 버퍼에서 첫 번째 색인 단위의 키 값이 2임을 찾을 수 있기 때문에 이것을 삭제하고 새로운 색인 단위를 버퍼에 삽입한다.

IBSF는 B트리에 데이터 삽입 시 색인 버퍼에서 중복된 색인 단위를 제거함으로써 색인 버퍼의 자원이 낭비되는 것을 막을 수 있기 때문에 커밋이 되는 시간을 지연시키고 결과적으로 NAND 플래시 메모리상에 B트리를 구축할 때 쓰기 연산의 횟수를 감소시킬 것이다.

3.3 삭제 정책

B트리 노드의 데이터 삭제가 발생했을 때 IBSF는 이것을 반영하기 위해 새로운 삭제타입의 색인 단위를 생성하고,




```
1.  delete(key) {
2.      삭제를 반영하는 색인 단위 A생성
        버퍼에서 중복된 색인 단위 검색
3.      if(중복된 색인 단위 B가 존재)
4.      {
5.        if(B가 삽입타입 색인 단위일 경우)
6.        {
7.          색인 단위 B를 버퍼에서 제거
8.        }
9.        if(B가 삭제타입 색인 단위일 경우)
10.       {
11.         B의 Flag set의 정보를 갱신
12.       }
13.     } else {
14.       삭제타입 색인 단위를 버퍼에 삽입
        }
15. }
```

(그림 4) IBSF의 삭제 정책

생성된 색인 단위는 색인 버퍼에 저장하기 때문에 삭제 정책은 삽입 정책이 처리되는 것과 유사하다. 그러나 다음과 같은 차이점이 있다. 첫째, 삭제를 반영하기 위한 색인 단위는 B트리 노드 내부에 삭제하고자 하는 엔트리의 위치정보만이 필요하다. 엔트리들의 위치정보는 일련의 비트열로 표현할 수 있기 때문에 하나의 삭제 타입의 색인 단위만으로 여러 개의 엔트리들의 삭제 정보를 유지할 수 있다. 예를 들어 (그림 4)에서 [1, 0, 0, 0, 0, 0]은 6개의 엔트리 중 첫 번째 엔트리를 삭제해야 한다는 정보를 의미한다. 둘째, 삭제 타입 색인 단위를 저장하려고 할 때 만약 삭제하고자 하는 엔트리와 동일한 키 값을 유지하고 있는 삽입타입 색인 단위가 존재할 경우 삭제할 데이터를 버퍼에 유지하고 있는 것은 자원 낭비이기 때문에 이것을 버퍼에서 제거한다. 예를 들어 (그림 4)에서 삭제 타입 색인 단위는 키 값이 7인 엔트리를 삭제하려고 하고 있고 색인 버퍼의 첫 번째 색인 단위는 키 값이 7인 삽입타입의 색인 단위이다. 따라서 IBSF는 이것을 제거하여 자원낭비를 예방한다. 세 번째, 만약 삭제 타입 색인 단위를 버퍼에 저장하려 할 때 동일한 노드 식별자를 지니고 있는 삭제타입 색인 단위가 버퍼에 존재할 경우 IBSF는 이 색인 버퍼에 새로운 삭제타입의 정보를 갱신한다. 예를 들어 (그림 4)의 색인 버퍼에서 에서 두 번째 색인 단위는 9번 노드에서 네 번째와 여섯 번째 엔트리를 제거하기 위한 정보를 저장하기 위해 [0, 0, 0, 1, 0, 1]의 비트열을 유지하고 있었고 새로 생성된 삭제 타입 색인 단위는 9번 노드의 첫 번째 엔트리를 제거하기 위해 비

트열 [1, 0, 0, 0, 0, 0]을 유지하고 있다. 따라서 새로운 색인 단위의 정보를 기존 색인 단위의 정보에 추가하면 [1, 0, 0, 1, 0, 1]이 되어 9번 노드의 삭제 정보를 유지하고 있는 삭제타입 색인 단위는 첫 번째, 네 번째, 여섯 번째 엔트리를 제거해야 한다는 정보를 유지하게 된다. 그러나 만약 동일한 노드에 대한 삭제타입 색인 단위가 존재하지 않는다면 색인 버퍼에 새로운 색인 단위를 저장하기 위한 자원을 할당해 준다. 이때 이미 색인 버퍼가 다른 색인 단위들로 다 채워져 있다면 커밋 연산에 따라 색인 버퍼가 유지하고 있는 색인 단위들을 플래시 메모리에 기록한다.

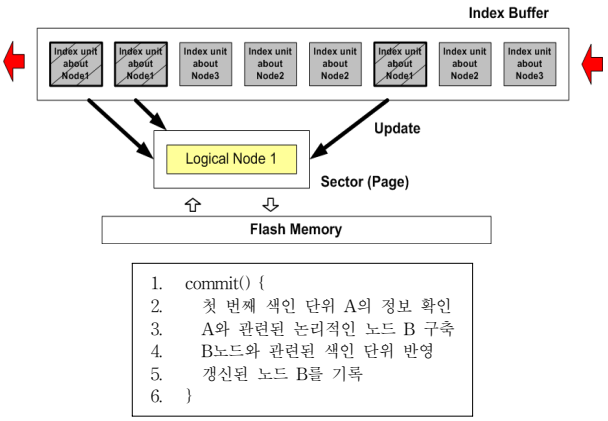
IBSF는 B트리에 데이터 삭제 시 색인 단위가 유지하고 있는 중복된 색인 단위를 제거하고 동일한 B트리 노드의 삭제 타입 색인 정보는 모아서 하나의 색인 단위로 처리하기 때문에 색인 단위의 개수를 줄일 수 있다. 이것은 버퍼가 채워지는 시간을 늦추기 때문에 B트리 구축 시 쓰기 연산의 횟수를 감소시킬 것이다.

3.4 커밋 정책

색인 버퍼가 일정 개수의 색인 단위들로 채워졌을 때 새로 생성된 색인 단위들을 유지할 공간을 확보하기 위해 IBSF는 색인 버퍼 내의 색인 단위들을 플래시 메모리로 커밋하여 빈 공간을 확보한다. 커밋 정책은 색인 버퍼 내의 색인 단위들을 커밋 할 때 사용하는 IBSF의 정책으로 선입선출(First In First Out, FIFO) 방식을 기반으로 동작한다. IBSF가 B트리 노드에 삽입이나 삭제를 반영하기 위해 새롭게 생성된 색인 단위들을 버퍼에 유지하려고 할 때 색인 버퍼가 이미 이전의 색인 단위들로 다 채워져 있다면 커밋 연산을 수행한다. 이때 색인 버퍼 내의 모든 색인 단위들을 커밋하는 것이 아니라 색인 버퍼에 가장 먼저 들어온 색인 단위를 확인하고 그 색인 단위 유지하고 있는 색인정보를 통하여 동일한 노드에 관련된 색인 단위들을 커밋 할 대상으로 선정한다. 예를 들어 (그림 5)와 같이 색인 버퍼가 다 채워졌을 때 색인 버퍼에 첫 번째로 삽입 되었던 색인 단위가 노드 1에 대한 색인 단위라고 가정하면 커밋할 대상이 되는 색인 단위들은 노드 1과 관련된 색인 단위들이다. 이 예제에서 노드 1에 관련된 색인 단위들은 첫 번째, 두 번째, 여섯 번째 색인 단위이다.

IBSF는 커밋 할 색인 단위들을 선정한 후 커밋 연산을 수행할 때 하나의 페이지에 모아서 저장하며, 색인 단위 형태로 저장하는 것이 아니라 논리적인 하나의 노드 형태로 구성한 후에 플래시 메모리에 기록한다. 따라서 하나의 논리적 B트리 노드를 구성하기 위해 이전에 커밋 되었던 페이지가 존재한다면 이것을 읽어온 후 커밋 대상으로 선정된 색인 단위들을 통해 새로운 색인 정보를 반영하여 한 개의 논리적인 B트리 노드를 구성한다. 그다음 이 논리적으로 구축된 B트리 노드를 플래시 메모리에 저장을 한 후 색인 버퍼에 이미 커밋이 끝난 색인 단위들을 제거함으로써 커밋 연산을 마친다.

(그림 5)의 예제에서 커밋 연산을 수행하기 위해 IBSF는 플래시 메모리로부터 노드 1의 데이터가 저장된 페이지를



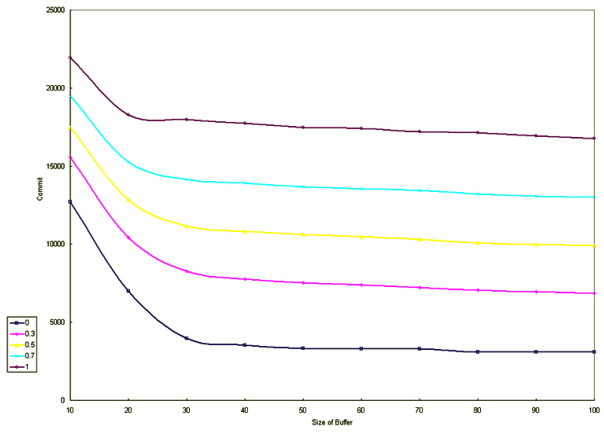
(그림 5) IBSF의 커밋 정책

읽어왔다. 그리고 색인 단위에 유지하고 있던 노드 1에 관련된 색인 단위들을 반영하여 논리적인 노드 1을 재구축한다. 색인 버퍼에는 논리적인 노드를 구축하기 위해 반영할 색인 단위들이 삭제 타입과 삽입타입 두 가지가 존재한다. 만약 삽입 타입을 먼저 반영한다면 노드 내에서 삭제해야 할 엔트리의 위치정보가 변경될 수 있음으로 색인 단위를 반영하는 순서는 삭제타입을 먼저 반영한 후 삽입타입을 반영해야 한다. 이렇게 구축된 논리적인 B트리 노드 1은 플래시 메모리로 다시 저장되며 노드 1을 구축하기 위해 사용되었던 색인 버퍼의 첫 번째, 두 번째, 여섯 번째 색인 단위들은 제거된다.

IBSF는 하나의 노드에 관련된 색인 단위들을 모아서 논리적인 노드 형태로 구축한 후 플래시 메모리의 하나의 페이지에 저장하기 때문에 추가적인 오버헤드를 발생시키는 노드 변환 테이블을 사용하지 않는다. 또한 저장된 B트리의 노드를 접근할 때 하나의 페이지를 읽는 것만으로 한 개의 B트리 노드를 접근할 수 있기 때문에 BFTL을 사용하는 것보다 더 좋은 읽기 성능을 보일 것이다.

4. 실험 및 결과

본 논문에서는 IBSF의 성능을 평가하기 위해 BFTL과 IBSF환경의 시뮬레이션을 구현하여 실험하였다. 실험에서 사용한 NAND 플래시메모리는 64 Mbytes의 크기로 설정하였고, 색인 버퍼의 크기는 (그림 6)의 실험결과를 통하여 결정하였다. (그림 6)의 X축은 색인 버퍼가 유지할 수 있는 색인 단위의 개수를 의미하고 Y축은 B트리가 구축이 될 때 커밋이 된 횟수를 의미한다. B트리를 구축하기 위해 24000개의 레코드를 삽입하였고 각 곡선은 B트리가 구축될 때 사용한 키 가 정렬된 비율(Ratio of sequence)을 의미한다. 0에 가까울수록 입력된 키의 순서가 순차적으로 키를 삽입한 것을 의미하고 1에 가까울수록 키의 순서는 랜덤한 순서로 사용하였다. (그림 6)의 결과에서 보여주는 것과 같이 30개의 색인 단위를 유지할 수 있는 크기의 색인 버퍼는 그 이상의 크기와 많은 차이가 없기 때문에 본 논문의 실험에서



(그림 6) 버퍼의 크기에 따른 커밋 횟수의 변화

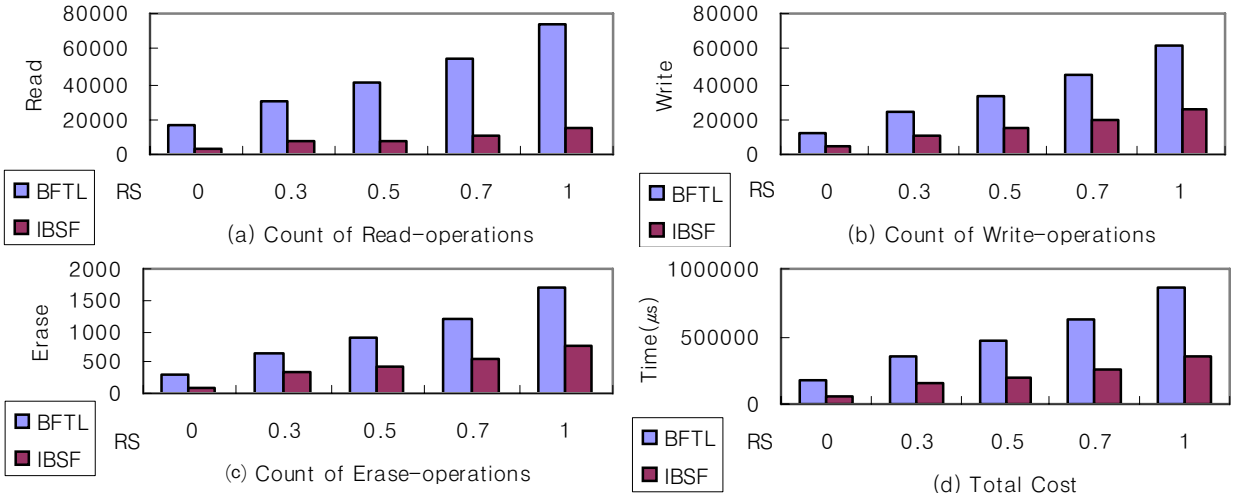
색인 단위의 크기는 30개로 설정하였다.

(그림 7)은 24000개의 레코드를 삽입하여 B트리가 구축될 때 BFTL과 IBSF의 성능을 측정한 결과이다. (그림 7)의 (a)부터 (d)의 X축인 RS(ratio of sequence)는 B트리를 구축하기 위해 삽입된 키 값의 순차적/랜덤 비율을 의미한다. 만약 RS가 0이면 B트리가 구축되는데 사용한 모든 레코드가 순차적인 순서로 삽입이 된 것이고 1이면 모든 레코드가 랜덤한 순서로 삽입된 것이다. 0.5일 때에는 절반은 순차적인 순서로 삽입하였고 나머지 절반은 랜덤한 순서로 삽입한 것을 의미한다. (그림 7)의 (a)부터 (b)의 Y축은 B트리가 구축될 때 발생하는 읽기, 쓰기, 지우기 연산의 횟수와 이 연산들의 횟수에 실제 플래시 메모리의 읽기, 쓰기, 지우기 연산의 가중치를 곱하여 B트리가 구축될 때 소모될 시간을 시뮬레이션 한 결과를 의미한다. 시뮬레이션에서 사용한 시간을 측정하는 구하는 식은 Time = (r × k1) + (w × k2) + (e × k3) 이고 r는 읽기 연산의 횟수, w는 쓰기 연산의 횟수, e는 지우기 연산의 횟수를 의미하고, k1, k2, k3는 각 연산에 대한 가중치를 의미한다. 플래시 메모리의 읽기, 쓰기, 지우기 연산이 처리되는데 필요한 시간적 비용은 소블록의 경우 각각 20㎲, 200㎲, 2㎳이다[15]. 본 실험에서는 플래시 메모리의 각 연산들이 1 : 10 : 100 비율의 시간적 비용이 소모됨으로 k1은 1, k2는 10, k3는 100을 넣어 시뮬레이션 하였다. (그림 7)의 실험 결과에서 볼 수 있듯이 키 값이 순차적일 경우 더 큰 차이로 IBSF의 성능이 BFTL보다 우수한 것을 볼 수 있다. 본 논문의 실험에서 RS가 0일 경우 IBSF는 BFTL보다 약 77.76% 읽기 연산의 성능을 개선할 수 있었고, 쓰기 연산의 성능은 약 66.21%, 지우기 연산의 성능은 약 74.3%, B트리가 구축될 때 소모되는 시간은 약 68.87% 개선된 것을 볼 수 있다. 본 논문의 실험의 실험을 통하여 NAND 플래시 메모리상에 B트리 구축 시 IBSF가 BFTL보다 더 좋은 성능을 보이는 것을 볼 수 있다.

5. 결 론

본 논문에서는 IBSF라고 불리는 새로운 색인 버퍼 관리 정책을 제안하였다. IBSF는 색인 버퍼에서 중복된 색인 단위를 제거하여 커밋 시간을 지연시켰고 이것은 결과적으로 플래시 메모리의 쓰기 및 지우기 연산의 수를 효과적으로 줄일 수 있다. 또한 IBSF는 추가적인 오버헤드를 가지고 있는 노드 변환 테이블을 제거하여 읽기 연산의 성능을 개선하였고 다양한 실험을 통하여 IBSF가 BFTL보다 좋은 성능을 보이는 것을 증명하였다.

본 논문에서 버퍼가 찼을 때 커밋을 할 색인 단위를 선택하기 위해 선입선출 기법을 사용하였다. 앞으로 NAND 플래시 메모리 하드웨어적 특징인 읽기, 쓰기, 지우기 연산 처리 시 소모되는 비대칭 비용을 고려하여 버퍼 교체 정책을 개발한다면 이것은 NAND 플래시 메모리에 B트리를 더 효율적으로 구축할 수 있을 것이다. 또한 예측하지 못한 전력손실 등으로 인한 시스템다운 시 색인 버퍼에 있는 데이터를 손실할 우려할 수 있고 이것은 컨커런시 컨트롤이나 시리얼라이즈 문제와 같은 다른 여러 가지 문제를 발생할 수 있기 때문에 이 문제를 해결하기 위한 정책 연구도 필요하다.



(그림 7) 시뮬레이션 결과

참 고 문 헌

[1] Tae-Sun Chung, Dong-Joo Park, Sangwon Park, Dong-Ho Lee, Sang-Won Lee, Ha-Joo Song, "System Software for Flash Memory: A Survey.", *International Conference on Embedded and Ubiquitous Computing*, pp394-404, 2006.

[2] D. S. Batory, "B+trees and indexed sequential files: a performance comparison," *ACM SIGMOD international conference on Management of data,* pp.30-39, 1981.

[3] Chin-Hsien Wu, Li-Pin Chang, Tei-Wei Kuo, "An Efficient B-Tree Layer for Flash-Memory Storage Systems," *Real - Time and Embedded Computing Systems and Applications* (RTCSA), pp.409-430, 2003.

[4] Sang-Won Lee, Dong-Joo Park, Tae-Sun Chung., Dong-Ho Lee, Sangwon Park, Ha-Joo Song, "A Log Buffer based Flash Translation Layer using Fully Associative Sector Translation," *ACM Transactions on Embedded Computing Systems* (accepted for publication).

[5] Intel Corporation, "Understanding the Flash Translation Layer(FTL) Specification." Technical report.

[6] Jeong-Uk Kang, Heeseung Jo, Jin-Soo Kim, and Joonwon Lee, "A Superblock-based Flash Translation Layer for NAND Flash Memory," *Proceedings of the 6th Annual ACM Conference on Embedded Software (EMSOFT 2006)*, pp. 161-170, 2006.

[7] Keun Soo Yim. et. al, "A Fast Start-Up Technique for Flash Memory Based Computing Systems," *ACM Symposium on Applied Computing*, pp.843-849, 2005.

[8] C. Bolchini. et. al, "PoLiDBMS: Design and Prototype Implementation of a DBMS for Portable Devices," *In Proc. Dodicesimo Convegno Nazionale sui Sistemi Evoluti per Basi di Dati, S. Margherita di Pula, I*, pp.71-76, 2004.

[9] Rajkumar Sen. et. al, "Efficient Data Management on Lightweight Computing Devices," *IEEE International Conference on Data Engineering*, pp.419-420, 2005.

[10] Keun Soo Yim. et. al, "A Flash Compression Layer for SmartMedia Card Systems," *IEEE Transactions on Consumer Electronics*, pp.192-197, 2004.

[11] ERAN GAL AND SIVAN TOLEDO, "Algorithms and Data Structures for Flash Memories," *ACM Computing Surveys*, Vol.37, No.2, pp.138-163. June 2005.

[12] Seung-Ho Lim and et. al, "An Efficient NAND Flash File System for Flash Memory Storage," *IEEE Transactions on Computers*, Vol.55, No.7, pp.906-912, July 2006.

[13] Sunhwa Park and Seong-Young Ohm, "New Techniques for Real-Time FAT File System in Mobile Multimedia Devices," *IEEE Transactions on Comsumer Electronics*, pp.1-9, 2006.

[14] Sunhwa PARK and et. al, Atomic Write FTL for Robust Flash File System, *IEEE International Symposium on Consumer Electronics*, pp.155-160, 2005.

[15] Samsung Electronics Company, K9F1208R0C 64M*8 bit NAND Flash-Memory Data Sheet, 2007.

이 현 섭

e-mail : hyunseob@cse.hanyang.ac.kr
2005년 천안대학교 컴퓨터공학과(학사)
2007년 한양대학교 컴퓨터공학과(석사)
2007년~현 재 한양대학교 컴퓨터공학과 박사과정
관심분야 : 데이터베이스, 플래시메모리용 시스템소프트웨어 등

주 영 도

e-mail : ydjoo@kangnam.ac.kr
1983년 한양대학교 전자통신공학과(학사)
1988년 미국 University of South Florida 컴퓨터공학과(석사)
1995년 미국 Florida State University 컴퓨터과학과(박사)
1984년~1985년 한국무역협회 전자계산소 시스템 프로그래머
1995년~2000년 KT 연구개발본부 연구팀/실장
2000년~2005년 시스코 시스템즈 코리아 통신사업부 담당 상무
2005년~2006년 화웨이 기술 코리아 부사장
2007년~현 재 강남대학교 컴퓨터미디어공학부 교수
관심분야 : 데이터베이스, 지능형시스템, 초고속정보통신망, 통신망관리 등

이 동 호

e-mail : dhlee72@hanyang.ac.kr
1995년 홍익대학교 컴퓨터공학과(학사)
1997년 서울대학교 컴퓨터공학과(석사)
2001년 서울대학교 컴퓨터공학과(박사)
2001년~2004년 삼성전자 책임 연구원
2004년~현 재 한양대학교 컴퓨터공학과 조교수
관심분야 : 데이터베이스, 멀티미디어 정보검색, 플래시메모리용 시스템소프트웨어 등