플래시 메모리 상에서 B-트리 설계 및 구현
(Design and Implementation of B-Tree on Flash Memory)

남 정 현 † 박 동 주 ††
(Jung-hyun Nam) (Dong-joo Park)

요 약 최근 PDA, 스마트카드, 휴대폰, MP3 플레이어와 같은 이동 컴퓨팅 장치의 데이타 저장소로 플래시 메모리를 많이 사용하고 있다. 이런 장치는 데이타를 효율적으로 삽입·삭제·검색하기 위해 B-트리와 같은 색인기법을 필요로 한다. 플래시 메모리 상에서의 B-트리 구현에 관한 기존 연구로서는 BFTL(B-Tree Flash Translation Layer) 기법이 최초로 제안 되었다. 플래시 메모리는 읽기연산보다 쓰기연산 비용이 훨씬 크며, 덮어쓰기(overwrite)가 불가능하다는 특징을 갖고 있다. 따라서 BFTL 기법에서는 B-트리 구축 시 발생되는 다량의 쓰기연산을 최소화하는데 초점을 맞추고 있다. 하지만 BFTL 기법에 성능 개선의 여지가 많이 남아 있으며, BFTL 기법이 SRAM 메모리 공간을 증가시킨다는 단점 때문에 비현실적이다. 본 논문에서는 플래시 메모리 상에서 효율적으로 B-트리를 구축하기 위한 BOF(B-Tree On Flash Memory)기법을 제안한다. BOF 기법의 핵심은, B-트리 구축 시 사용하는 임시 버퍼의 인덱스 유닛(index unit)들을 플래시 메모리에 저장할 때 같은 노드에 속하는 인덱스 유닛들을 같은 섹터에 저장하는 것이다. 본 논문에서는 성능평가 실험을 통해 BOF 기법의 우수성을 보인다.

키워드 : 플래시 메모리, B-트리, 플래시 변환 계층

Abstract Recently, flash memory is used to store data in mobile computing devices such as PDAs, SmartCards, mobile phones and MP3 players. These devices need index structures like the B-tree to efficiently support some operations like insertion, deletion and search. The BFTL(B-tree Flash Translation Layer) technique was first introduced which is for implementing the B-tree on flash memory. Flash memory has characteristics that a write operation is more costly than a read operation and an overwrite operation is impossible. Therefore, the BFTL method focuses on minimizing the number of write operations resulting from building the B-tree. However, we indicate in this paper that there are many rooms of improving the performance of the I/O cost in building the B-tree using this method and it is not practical since it increases highly the usage of the SRAM memory storage. In this paper, we propose a BOF(the B-tree On Flash memory) approach for implementing the B-tree on flash memory efficiently. The core of this approach is to store index units belonging to the same B-tree node to the same sector on flash memory in case of the replacement of the buffer used to build the B-tree. In this paper, we show that our BOF technique outperforms the BFTL or other techniques.

Key words : Flash Memory, B-Tree, FTL

1. 서 론

최근 들어 컴퓨팅 분야의 기술발전이 유비쿼터스 기반의 컴퓨팅으로 변환됨에 따라 휴대폰, PDA, 스마트카드, 디지털 카메라와 같은 이동 컴퓨팅 장치의 사용이 급증하면서 플래시 메모리에 대한 관심이 증가하고 있다. 플래시 메모리는 가볍고 물리적인 충격에 강해 휴대가 용이하며 저전력으로 동작해 배터리 소모량을 줄이기 때문에 하드디스크를 대체할 매체로 각광받고 있다[1,2]. 하지만 플래시 메모리는 빠른 읽기연산 속도에 비해 쓰기연산 속도가 다른 메모리보다 느리고 하드디스크처럼 덮어쓰기(overwrite)연산이 지원되지 않는 단점을 가지고 있다[2].

플래시 메모리 상에서 덮어쓰기를 수행하기 위해서는 먼저 해당 블록(block)에 대하여 플래시 메모리의 연산 중 가장 비용이 많이 요구되는 삭제(erase)연산을 수행해야만 한다. 이러한 문제점을 극복하기 위해 플래시 메

· 본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음
† 정 회 원 : 숭실대학교 컴퓨터학부
jhnam77@hanwha.co.kr
†† 정 회 원 : 숭실대학교 컴퓨터학과 교수
djpark@computing.ssu.ac.kr
논문접수 : 2006년 1월 25일
심사완료 : 2006년 11월 27일

모리는 파일 시스템과 플래시 메모리 사이에 위치하는 플래시 변환 계층(Flash memory Translation Layer) 이란 시스템 소프트웨어를 사용한다[3,5-8]. 플래시 변환 계층은 다양한 블록교체기법을 적용하여 사용자에게 비교적 수행시간이 오래 걸리는 삭제연산에 대하여 이미 삭제연산을 수행한 빈 블록으로 재사상(re-mapping) 함으로써 플래시 메모리를 기존의 하드디스크와 같이 사용할 수 있도록 해 준다.

플래시 메모리의 사용이 증가함에 따라 플래시 메모리에 저장된 데이타를 효율적으로 관리하기 위한 해시 테이블(hash table), 검색 트리(search tree)와 같은 색인기법들이 필요하다. 특히, B-트리 색인 구조는 데이타의 효율적인 삽입·삭제·검색이 용이하며 확장성이 우수한 색인기법으로 유명하다. 하지만 하드디스크에서와 달리 플래시 메모리 상에서 B-트리 구현은 빈번한 쓰기 연산으로 인한 덮어쓰기가 증가하여 많은 구축비용을 소비한다는 것이다. 따라서 플래시 메모리 환경에 적합한 B-트리 구현기법이 필요하다.

플래시 메모리 상에서 B-트리를 구현하기 위해 기존 연구[9]에서 BFTL(B-Tree Flash Translation Layer: 이하 BFTL) 기법이 제안되었으나 거대한 사상 테이블을 유지하기 위한 별도의 SRAM이 필요하며, B-트리 검색 시 읽기연산이 많이 발생하여 검색속도가 오래 걸리는 단점을 가지고 있다. 본 논문에서는 플래시 메모리 상에서 효율적으로 B-트리를 구축하기 위한 BOF(B-Tree On Flash memory) 기법을 제안한다. BFTL 기법과는 달리, BOF 기법에서는 B-트리 구축 시 사용하는 유보버퍼(reservation buffer)의 인덱스 유닛(index unit)들을 플래시 메모리에 저장할 때, 같은 B-트리 노드에 속하는 인덱스 유닛들을 같은 섹터(sector)에 저장한다. 이로써 BOF 기법에서는 거대한 사상 테이블을 필요로 하지 않으며 또한 B-트리에 대한 키 검색 시 사상 테이블을 이용하지 않음으로써 검색 비용을 줄일 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 플래시 메모리의 특성에 대해 논의한다. 3장에서는 기존 연구인 BFTL 기법과 단점에 대해 기술한다. 4장에서는 본 논문에서 제안하는 플래시 메모리상에서 효율적인 B-트리 설계 및 구현을 위한 BOF 기법에 대해 기술한다. 5장에서는 실험 결과 및 BOF 기법의 우수성을 보인다. 마지막으로 6장에서 결론을 맺는다.

2. 플래시 메모리 특성

'플래시 메모리는 전력 소모가 적고 충격에 강하며 전원이 꺼지더라도 저장된 데이타가 사라지지 않는 비휘발성 메모리 특성을 가지고 있다. 이러한 특성으로 인해 최근 들어 이동 컴퓨팅 시스템의 저장장치로 널리 사용되고 있다. 본 장에서는 플래시 메모리의 특성에 대하여 간략히 기술한다.

플래시 메모리는 크게 바이트(byte)단위 입·출력을 지원하는 NOR형과 섹터(sector)단위 입·출력만 지원하는 NAND형으로 구분된다. NOR형 플래시 메모리는 읽기연산의 속도가 빠른 반면 쓰기연산의 속도가 느려 주로 프로그램 코드용 메모리로 사용하며[4], NAND형 플래시 메모리는 NOR형 플래시 메모리와는 달리 쓰기 연산의 속도가 빠르고 단위 공간당 단가가 낮아 주로 대용량 데이타 저장장치로 사용된다[11]. 본 논문에서는 대용량 저장장치로 많이 사용되는 NAND형 플래시 메모리를 사용하여 플래시 메모리 상에서 B-트리 설계 및 구현에 관하여 주로 살펴본다.

플래시 메모리는 다수개의 블록(block)으로 구성되며 그림 1과 같이 하나의 블록에는 32개의 페이지(page)가 포함되며 하나의 페이지는 512 바이트의 데이타를 저장하는 데이타 영역(섹터)과 16 바이트의 예비 영역으로 구성된다[5]. 하드디스크와 달리 데이타가 저장된 섹터에 대해 덮어쓰기가 허용되지 않기 때문에 플래시 메모리에 데이타를 덮어쓰기 위해서는 저장된 데이타에 대한 삭제연산이 선행되어야 하며 삭제연산은 읽기·쓰기 연산 단위인 페이지단위로 수행되지 않고 블록단위로 수행되기 때문에 표 1과 같이 수행시간이 오래 걸리는 연산으로 빈번한 삭제연산의 발생은 플래시 메모리 성능저하의 원인이 된다[1,2]. 이러한 문제점 때문에 플래시 메모리와 호스트 시스템의 파일시스템 중간에 위치하는 플래시 변환 계층을 사용하여 해결한다[3,5-8].

플래시 변환 계층은 플래시 메모리를 보다 효율적으로 사용하기 위한 다양한 블록교체기법을 사용한다. [5-8]에서의 블록교체기법은 덮어쓰기가 발생하면 미리 할당된 여유블록(spare block)에 기존 블록의 유효한(valid) 데이타와 갱신할 데이타를 복사하고 이를 연결리스트로 구성해 읽기연산이 수행될 때 역순으로 검색하여 가장 최신의 데이타를 제공하는 방법이다. 이 같은 방법을 제공하기 위해, 플래시 변환 계층은 주소 사상

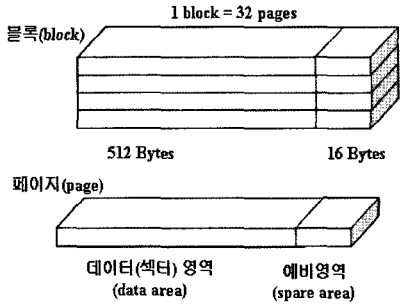

그림 1 플래시 메모리 구조

표 1 다양한 기억장치 간의 성능비교

	접근시간		
	읽기 (Read)	쓰기 (Write)	삭제 (Erase)
DRAM	60ns(2B) 2.6μs(512B)	60ns(2B) 2.6μs(512B)	-
NOR형 플래시	150ns(1B) 15μs(512B)	211μs(2B) 3.5ms(512B)	1.2s(128KB)
NAND형 플래시	10μs(1B) 36μs(512B)	226μs(2B) 266μs(512B)	2ms(16KB)
하드디스크	12.4ms(512B)	12.4ms(512B)	-

테이블(address mapping table)을 사용하여 파일시스템으로부터 할당된 논리적 주소를 플래시 메모리의 물리적 주소로 변환시켜 재사상 함으로써 덮어쓰기 발생 시 삭제연산을 감출 수 있어 플래시 메모리의 쓰기연산 성능을 개선했다. 하지만 플래시 메모리의 쓰기연산은 다른 저장장치와 비교해볼 때 상대적으로 느리기 때문에 여전히 성능을 개선할 필요가 있다. 이와 같은 플래시 변환 계층의 한계성으로 인해 직접 플래시 메모리상에서 하드디스크와 동일한 구현기법으로 B-트리를 구현하는 것은 구축 시 상당한 오버헤드를 발생시킨다.

3. BFTL 기법의 개념 및 장단점

기존 하드디스크와는 달리, 플래시 메모리상에서 B-트리를 구현할 때에는 플래시 메모리로의 쓰기연산을 최대한 감소시키는 것이 중요하다. 본 장에서는, 플래시 메모리상에서 B-트리를 구축하기 위해 기존 연구에서 최초로 제안된 BFTL 기법[9]에 대해 기술한다.

3.1 BFTL 기법의 개요

기존 하드디스크 상에서의 B-트리 설계 및 구현 기술을 그대로 플래시 메모리에 적용하는 것은 상당한 쓰기연산을 유발시켜 B-트리 구축 성능을 저하시킨다. 이를 개선하기 위해 BFTL 기법이 최초로 제안되었으며[9], 그 아이디어는 다음과 같다. B-트리에 어떤 키(key)가 삽입되었을 때, 하드디스크에서는 해당 단말노드(leaf node)에 대응되는 페이지를 찾은 후 그 페이지에 키를 저장하면 된다. 이때 플래시 메모리 관점에서 보면 키의 삽입은 해당 페이지에 대한 덮어쓰기가 되며, 전체적인 덮어쓰기의 수를 줄이는 게 중요하다. 이를 위해서, 매 키의 삽입 시에 해당 단말노드에 바로 저장하는 것이 아니라 유보버퍼(reservation buffer)에 키에 대한 정보를 인덱스 유닛(index unit)[1] 형태로 임시적으로 저장한다. 이로써 매 키 삽입에 따른 덮어쓰기를 회피할 수 있다. 만약 유보버퍼가 다 차게 되면, 일정 수의 인덱스 유닛들을 플래시 메모리의 한 페이지에 한꺼번에 저장한다. 이때, 서로 다른 B-트리 노드에 속하는 인덱스 유닛들이 같은 페이지에 저장될 수 있으므로, 이를 추적하기 위한 노드 변환 테이블(Node Translation Table)을 유지한다. 여기에는 각 B-트리 노드에 속하는 키들이 플래시 메모리에 저장되어 있는 위치 정보가 저장되어 있다.

3.2 BFTL 기법의 동작

BFTL 기법의 구성은 새로 입력된 인덱스 유닛을 임시로 저장하여 쓰기연산을 최대한 지연시키기 위한 유보버퍼, 플래시 메모리에 흩어져 저장되어있는 인덱스 유닛의 논리적 섹터 번호를 기록한 노드 변환 테이블, 유보버퍼 내에 있는 인덱스 유닛을 플래시 메모리로 저장하기 위한 수용정책으로 구성된다.

그림 2는 BFTL 기법 구현을 도식화한 그림이다. B-트리와 관련된 응용프로그램에 의해 키 값이 삽입되면 단말 노드를 검색하여 해당 노드에 대한 인덱스 유닛을 생성한다. 생성된 인덱스 유닛은 일시적으로 유보버퍼에 적재되며, 만약 유보버퍼 내에 빈 공간이 없을 경우 수용정책에 의해 유보버퍼에 적재된 인덱스 유닛을 플래시 메모리의 새로운 섹터를 할당하여 저장하는 작업을 수행한다.

수용정책에 의해 인덱스 유닛을 플래시 메모리로 저장할 때 그림 2에서처럼 유보버퍼 내 적재된 순서에 따라 플래시 메모리 섹터에 저장하기 때문에 유보버퍼 내에는 서로 다른 노드에 속하는 인덱스 유닛들이 동일한 섹터에 저장될 수 있다. 따라서 플래시 메모리상에 흩어진 인덱스 유닛들을 관리하기 위하여 SRAM과 같은 휘발성 메모리에 그림 3과 같은 노드 변환 테이블을 구성한다. 노드 변환 테이블은 노드를 구성하는 인덱스 유닛이 저장된 플래시 메모리의 논리적 섹터 번호를 연결리스트로 연결한 테이블이다. 예를 들어, 그림 3에서 C 노드는 플래시 메모리의 논리적 섹터 번호 11, 36, 21 섹터를 읽어 노드를 구성한다. 따라서 노드를 구성하기 위한 읽기연산이 증가하며 노드 변환 테이블을 유지하기 위한 메모리 공간을 요구한다는 단점과 노드 변환 테이블의 노드 리스트가 무한적 증가할 수 있다는 문제점도 발생한다. 이와 같은 문제점을 극복하기 위해 BFTL 기법은 노드 변환 테이블의 최대 리스트 수를 설정하여, 만약 임의의 노드를 구성하기 위한 노드 변환 테이블의 리스트가 최대 리스트 수를 초과하면 리스트의 논리적 섹터 번호에 해당하는 섹터를 읽어 노드와 관련 있는 최신의 인덱스 유닛을 추출하여 새로 할당된 섹터에 압축하는 리스트 압축을 수행한다.

1) 인덱스 유닛은 B-트리의 엔트리(entry)에 포함되는 정보와 이러한 정보가 플래시 메모리 상의 어디에 저장되었는지에 대한 위치 정보를 포함한다. 이후 동일한 인덱스 유닛을 사용하는 BOF 기법 설명에서 자세히 다룬다.

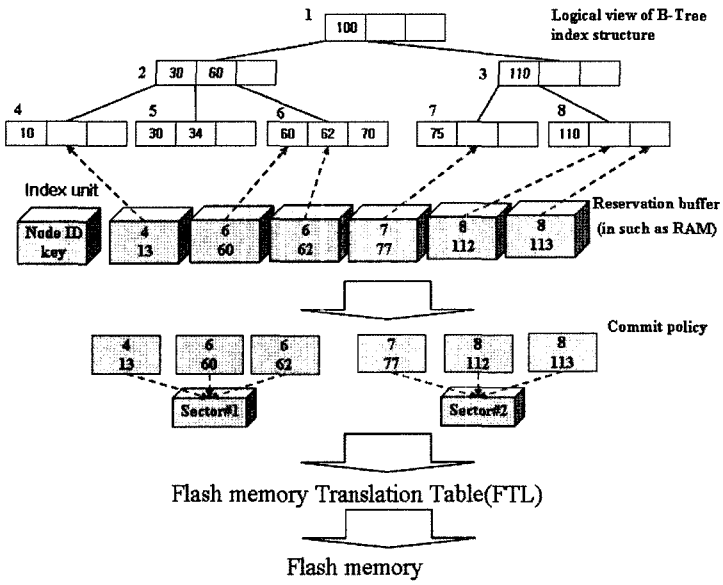

그림 2 BFTL 기법의 동작 원리

(a) Logical B-Tree (b) Note Translation Table

그림 3 노드 변환 테이블

3.3 BFTL 기법의 문제점

위에서 설명한 BFTL 기법이 갖는 문제점은 다음과 같다. 첫째, BFTL 기법에서는 덮어쓰기에 잘 대처하지 못하는, 가장 단순한 블록사상(Block mapping) FTL 기법을 가정하고 있으므로 현실성이 결여되어 있다. 둘째, BFTL 기법에서는, B-트리를 구성하는 키의 수가 커지면 노드 변환 테이블도 커지게 되어 SRAM 메모리가 더 많이 필요하고 이는 하드웨어 구성비용의 증가를 가져오며 휴대용 저장장치와 같은 매체에 사용이 불가하다[14]. 셋째, 키 검색을 위해 B-트리 노드 순환(traverse)을 할 때, 한 노드가 플래시 메모리의 여러 페이지에 저장되어 있으므로 한 노드를 구성하기 위한 페이지 읽기연산이 많이 유발된다. 따라서 키 검색 성능이 떨어진다.

위와 같은 문제점들을 해결하기 위해서, 본 논문에서 BOF 기법을 제안한다. BOF 기법에서는, 뛰어난 FTL 기법의 적용과 BFTL 기법과 다른 인덱스 유닛 쓰기 정책만으로도 효율적으로 B-트리 구축과 검색을 할 수 있음을 보인다.

4. BOF 기법을 이용한 B-트리 구축

플래시 메모리 상에서 효율적으로 B-트리를 구축하기 위해 중요한 고려사항은 키 값 삽입 시 덮어쓰기 횟수를 최대한 감소시키는 것에 있다. BOF 기법은 BFTL과 동일한 인덱스 유닛, 유보버퍼를 사용하여 덮어쓰기를 감소시켰으며 한 섹터 당 한 노드를 저장하는 독특한 방법을 사용하여 효율적인 노드관리에 용이하며 BFTL 기법의 문제점인 검색 시 읽기연산을 감소시켰다. 본 장에서는 플래시 메모리 환경에서 효율적으로 B-트리를 구현하기 위해 본 논문에서 제안하는 BOF 기법에 대해 기술한다.

4.1 BOF 기법의 구성

하드디스크 상에서 B-트리를 구현하기 위해서는 노드 단위(4KB)의 I/O가 빈번히 발생하지만 읽기·쓰기연산

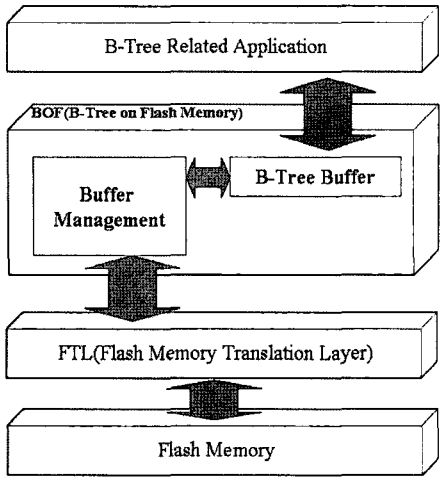

그림 4 BOF 기법 구성

에 대하여 동일 I/O 비용으로 B-트리를 구현할 수 있다[12]. 하지만 하드디스크 상의 구현기법과 동일방법을 사용하여 플래시 메모리상에 B-트리를 구현하는 것은 플래시 메모리의 물리적인 특성으로 인해 심각한 성능 저하를 초래한다. 따라서 플래시 메모리에 특성을 고려한 B-트리 구현기법이 필요하다.

BOF 기법은 B-트리 관련 응용프로그램 계층과 플래시 변환 계층 중간에 위치하기 때문에 기존 응용프로그램의 수정 없이 구현이 가능한 시스템 소프트웨어이다. 그림 5는 BOF 기법 구성을 도식화한 그림으로 인덱스 유닛을 일시적으로 저장하는 유보버퍼, 유보버퍼의 인덱스 유닛을 플래시 메모리로 저장하는 버퍼 관리로 구성된다. 이러한 BOF 기법을 사용하여 플래시 메모리 상에서 효율적으로 구현할 수 있으며 BFTL 방법과 다른 기법을 사용하여 다양한 블록교체기법에 적용이 가능하다.

4.2 인덱스 유닛(Index Unit)

BOF 기법은 B-트리 노드를 구성하기 위해 BFTL 기법과 유사한 인덱스 유닛을 사용한다. 그림 6의 인덱스 유닛은 B-트리 노드의 각 엔트리 정보를 저장하는 단위를 말하며 다수개의 인덱스 유닛이 모여 하나의 B-트리 노드를 구성한다. 인덱스 유닛은 각 노드마다 유일하게 부여된 노드 번호 및 각 노드가 플래시 메모리에 저장될 논리적 섹터 번호를 기록하는 노드 식별자(node Id), 상위노드의 노드 번호를 가리키는 부모 노드 지시자(parent node pointer)와 좌·우 자식 노드의 노드 번호를 가리키는 좌·우 자식 노드 지시자(left·right node pointer), 인덱스 유닛이 유보버퍼에 삽입된 시간을 기록하는 시간 스탬프(time stamp), 단말 노드와 비단말 노드를 식별하는 단말 플래그(leaf flag)등으로 구성된다. 이러한 인덱스 유닛의 크기(21Byte)는 하드디스크 상의 B-트리 노드 크기(4KB)에 비해 크기가 작기 때문에 인덱스 유닛을 저장하기 위한 적은 공간의 유보버퍼를 사용할 수 있으며 빈번히 플래시 메모리로 쓰기 연산을 수행하는 인덱스 유닛의 중복을 방지할 수 있다. BOF 기법은 BFTL와 달리 검색비용을 줄이고 효율적인 노드관리를 위해 같은 노드에 속하는 인덱스 유닛들을 플래시 메모리상의 동일한 섹터에 저장한다. 즉, 한 섹터에 한 노드를 저장함으로써 하드디스크 상에서 구현한 방법과 같이 특정 노드를 구성할 때 한 번의 읽기 연산을 수행하여 노드를 구성할 수 있다.

4.3 버퍼 관리(Buffer Management)

BOF 기법은 덮어쓰기를 최대한 지연시키기 위해 갱신된 데이타(dirty data)를 임시로 저장하는 유보버퍼를 사용한다. 유보버퍼는 생성된 인덱스 유닛을 임시로 저장하여 빈번히 참조되는 인덱스 유닛에 대한 플래시 메모리로의 덮어쓰기를 감소시켰다. 하지만 SRAM과 같은 휘발성 메모리에 구현되는 유보버퍼의 크기는 한정된다. 따라서 B-트리 노드에 대한 변경정보를 반영하는 인덱스 유닛을 일시적으로 저장하고 있는 유보버퍼는 일정 간격으로 버퍼안의 데이타를 비우는 작업을 수행한다. 이때 버퍼 관리는 유보버퍼의 크기가 초과되면 버퍼안의 데이타를 비우기 위해 희생자(victim)를 선정한다. 희생자 선정 기준은 유보버퍼의 가장 앞쪽에 자리하고 있는 인덱스 유닛을 희생자로 선정하여 버퍼 내에서 동일한 노드 식별자를 갖는 인덱스 유닛을 압축하여 플래시 메모리에 저장한다. 또한 노드 분리(node split)가 발생하면 플래시 메모리에 이전 섹터에 저장된 유효하지 않은 데이타를 제거하고 최신의 노드 데이타를 저장하기 위해 플래시 메모리로 쓰기연산을 수행한다. 이와 같이 버퍼 관리는 한정된 유보버퍼의 크기를 초과하거

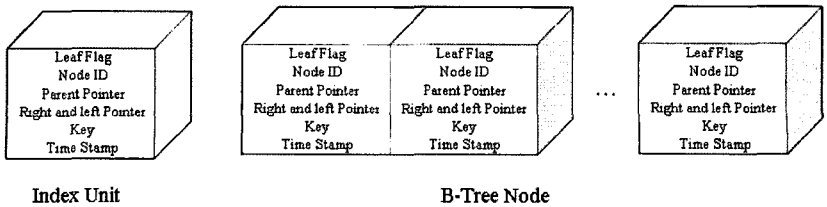

그림 5 인덱스 유닛(Index Unit)

나 노드 분리(node split)가 발생하면 동일 노드 식별자를 갖는 인덱스 유닛을 플래시 메모리의 동일 섹터에 저장하는 작업을 말한다. 이와 같이 버퍼 관리를 사용하여 플래시 메모리에는 항상 최신의 데이타가 저장되는 장점을 갖는다.

4.4 BOF 기법의 동작

BOF 기법은 B-트리 구축 시 사용하는 유보버퍼의 인덱스 유닛을 플래시 메모리에 저장할 때 같은 노드에 속하는 인덱스 유닛들을 같은 섹터에 저장하는 것이다. 이로써 BFTL에서 사용한 노드 변환 테이블을 저장하기 위한 메모리가 필요 없으며 검색 비용도 감소한다.

그림 6은 BOF 기법의 구현을 도식화한 그림이다. 그림 6에서처럼 입력된 데이타를 노드에 삽입하기 위해서는 삽입할 단말 노드를 루트 노드(*root node*)로부터 검색하고 인덱스 유닛을 생성한다. 이와 같이 단말 노드를 검색하기 위해서는 루트 노드부터 단말 노드까지의 각 단계별 노드를 구성해야 한다. 따라서 [13]의 방법과 달리 플래시 메모리에 이미 저장된 노드와 유보버퍼에 적재되어 있는 인덱스 유닛을 그림 7과 같이 병합하는 노드 재구성(node reorganization)작업을 수행한다. 예를 들어 그림 7에서처럼 노드 식별자가 5인 노드를 구성한다고 가정하면 플래시 메모리상에 이미 저장된 5번 노드에 해당하는 섹터를 읽어 유보버퍼에 상주하고 있는 최신의 인덱스 유닛을 사용하여 노드를 재구성함으로써 삽입 및 검색 시 필요한 노드를 구성할 수 있다. 이처럼 노드 재구성 작업을 통해 얻어진 단말 노드가 인덱스 유닛을 삽입할 빈 공간이 없다면 노드 분리(node split)가 발생한다. 노드 분리가 발생하면 BOF 기법은 버퍼 관리에 의해 기존 노드와 새로 생성된 노드를 유보버퍼가 아닌 플래시 메모리의 노드 식별자에 해당하는 섹터

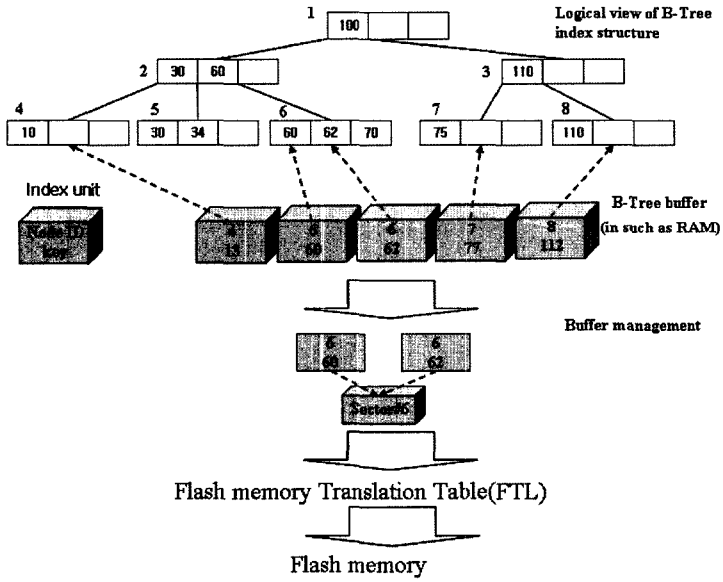

그림 6 BOF 기법의 동작 원리

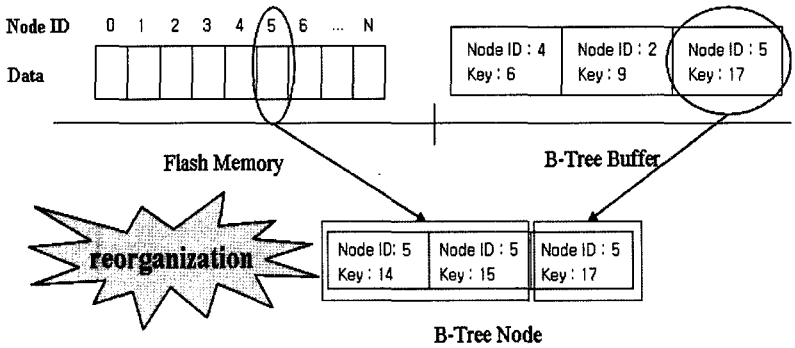

그림 7 BOF에서의 노드 재구성

```
insert(Key K, Tid T) {
  find the leaf node N to store K;   /* traverse the B-tree starting from the Root node*/
  create an index unit I to insert K and T into;
  if(N is not full) {
    write I to the B-tree reservation buffer;  /* this buffer is used to store index units temporarily*/
  }
  else {
    splitnode(N, I);  /* the leaf node N is full */
  }
}
splitnode(Node N, IndexUnit I) {
  create a new node N' and distribute some keys in N to N';
  write N and N' to flash memory directly;
  find the parent node P using parent_node_ptr of I;
  if(the parent node is not full) {
    insert the new index key into P;
    for each entry in P, store it in the form of the index unit to the reservation buffer;
  }
  else {
    create a new index unit I' and insert the new index key into I';
    splitnode(P, I');  /* process propagation upward */
  }
}
```

그림 8 BOF 알고리듬

에 바로 저장한다. 노드 분리 후 유보버퍼에 적재하지 않고 플래시 메모리에 바로 저장함으로써 노드 분리가 수행되기 전에 플래시 메모리에 저장되어 있던 기존 노드의 유효하지 않은 데이타가 플래시 메모리에 저장되는 것을 방지할 수 있다. 따라서 플래시 메모리에는 최신의 노드 데이타가 저장되며 노드 구성 시 유효한 데이타를 사용하여 노드를 구성할 수 있다. 반면, 재구성된 노드에 인덱스 유닛을 삽입할 여유 공간이 있으면 재구성된 노드에 새로 입력된 인덱스 유닛을 삽입하여 B-트리 노드를 구성한다. 이처럼 생성된 인덱스 유닛을 유보버퍼에 적재함으로써 인덱스 유닛이 플래시 메모리에 덮어쓰기 되는 것을 방지할 수 있고 필요시 플래시 메모리에 노드 데이타를 바로 저장함으로써 최신의 노드 정보를 유지할 수 있는 장점을 갖는다. 그림 8은 앞서 말한 내용을 알고리듬화한 것이다.

이와 같이 BOF 기법은 한 섹터 당 한 노드를 저장하여 한 번의 읽기연산으로 원하는 노드를 구성할 수 있으므로 검색 시 읽기연산을 감소시켰으며 효율적인 노드관리에 용이하다. 또한 노드 변환 테이블을 유지하기 위한 추가적 메모리 공간이 요구되지 않는다. 따라서 이동 컴퓨팅 장치와 같은 메모리의 한계가 있는 매체에서도 저비용으로 구현이 가능하다는 장점을 갖는다.

결과적으로 BOF 기법은 B-트리를 구축할 때 자주 참조되는 데이타를 유보버퍼에 저장하여 덮어쓰기를 감소시켰으며, BFTL의 문제점을 극복하여 보다 효율적인 방법으로 B-트리를 구현하였다.

5. 실험 결과

본 장에서는 플래시 메모리 변환 계층의 다양한 블록교체기법을 이용하여 BOF 기법을 적용한 결과와 적용하지 않은 결과에 대하여 실험을 통한 성능분석을 수행하여 본 논문에서 제안하는 BOF 기법의 우수함을 규명한다.

성능 평가 결과는 B-트리 구축 시 발생한 각 연산(읽기, 쓰기, 삭제)의 비용을 구하고 이를 합산하여 총비용으로 나타낸다.

그림 9는 5,000 개의 랜덤 키 값을 삽입하여 B-트리를 구축했을 때의 BOF와 BFTL 기법의 구축비용을 성능평가한 그림이다. 그림 9에서 알 수 있듯이 BFTL 기법의 경우 비교적 성능이 우수한 블록교체기법인 로그 블록 스킴을 사용하더라도 구축 시 덮어쓰기를 수행하지 않고 항상 새로운 섹터에 데이타를 저장하고 일정 간격으로 쓰레기 수집을 수행하기 때문에 동일한 구축 비용이 요구된다. 하지만 BOF의 경우 효율적으로 덮어쓰기를 제어하는 우수한 블록교체기법을 사용하면 구축 비용이 점차 감소함을 알 수 있다. 그림 10은 비교적 성능이 우수한 블록교체기법을 사용하여 랜덤 키 값 1,000 개를 검색했을 때의 BOF와 BFTL의 읽기연산 횟수를 나타낸 그림으로 노드 변환 테이블을 사용하여 다수의 읽기연산을 통해 노드를 구성하는 BFTL에 비해 한 번의 읽기연산으로 노드를 구성하는 BOF 기법이 읽기연산횟수가 감소하는 것을 알 수 있다. 이처럼 BOF 기법은 BFTL 기법의 문제점을 극복하였으며 우수한 블록교체기법을 사용하면 보다 뛰어난 성능을 나타냄을 보였다. 하지만 BFTL 기법과 BOF 기법의 구조는 유사하나 이를 직접적으로 비교하는 것은 그 구현 방법이

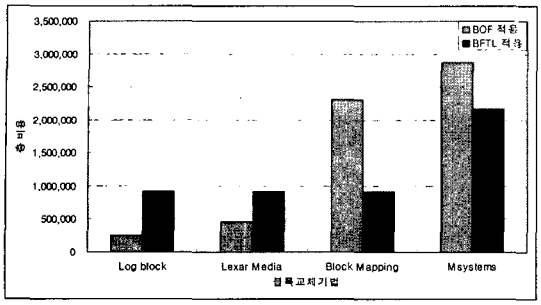

그림 9 B-트리 구축 시 BOF와 BFTL 기법 성능평가

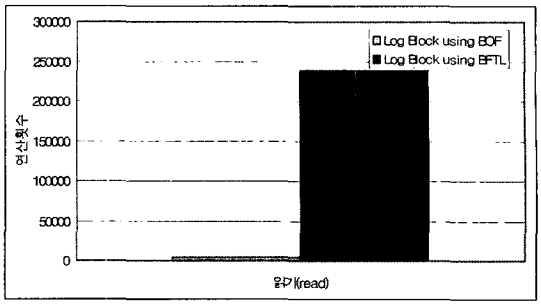

그림 10 B-트리 검색 시 BOF와 BFTL 기법 성능평가

서로 상이하여 무의미 하다고 판단된다. 따라서 본 논문에서는 플래시 변환 계층에서의 B-트리 구축과 BOF 기법을 사용한 B-트리 구축에 대하여 성능평가를 수행하였다.

플래시 메모리 상에서 B-트리 구축은 삽입연산에 의해 구현된다. 이때, 실제 BOF 기법에서 한 노드가 가질 수 있는 최대 엔트리 수는 21 바이트의 크기를 갖는 인덱스 유닛 25개를 저장할 수 있으나 본 논문의 실험에서는 최대 엔트리 수를 7로 하여 B-트리의 깊이(depth)를 증가시켰으며, 유보버퍼의 크기를 30(1KB)부터 480(12KB)까지 설정하고 버퍼 크기 증가에 따른 성능평가를 수행하였다. 본 논문에서 사용한 플래시 메모리 장치로는 64MB NAND형 플래시 메모리를 사용하며 실험을 위한 데이타는 10,000개의 랜덤 데이타를 이용하여 B-트리를 구축하였다. B-트리 검색실험은 랜덤 데이타 5,000개를 이용하여 수행된 읽기연산의 횟수를 계산한다.

그림 11은 유보버퍼의 크기를 30으로 설정하고 키 값을 삽입 하여 플래시 메모리의 각 연산의 서로 다른 상대비용[1]을 적용한 결과이다. 그림 11에서처럼 B-트리에 키 값이 삽입되면 BOF 기법을 적용한 블록교체기법의 경우 유보버퍼를 사용하여 빈번히 참조되는 인덱스 유닛이 중복해서 플래시 메모리로 저장하는 것을 감소시켜 덮어쓰기가 감소하여 보다 우수한 성능을 나타낸다.

그림 12는 B-트리 검색 시 수행된 읽기연산의 횟수를 도식화하여 나타낸 그림이다. 그림 12에서 알 수 있듯이 BOF 기법의 경우 루트 노드(root node)로부터 검색하지 않고 유보버퍼의 인덱스 유닛을 먼저 검색하여 B-트리 검색 깊이(depth)를 감소시켰다. 즉, B-트리 검색이 루트 노드로부터 수행되지 않고 유보버퍼에서 얻어진 인덱스 유닛의 하위 노드부터 수행하기 때문에 읽기연산이 감소함을 알 수 있다. 이와 같은 실험결과를 통해 상대적으로 수행시간이 오래 걸리는 덮어쓰기를 감소시켜 플래시 메모리 상에서 B-트리를 구축할 때 구축비용을 감소시켰다.

그림 13은 비교적 성능이 우수한 블록교체기법인 로그 블록교체기법[5]에 BOF 기법을 적용해서 유보버퍼 크기가 증가함에 따른 구축 시 총비용과 검색 시 수행된 읽기연산의 횟수를 나타낸 것이다. 그림 13에서처럼 유보버퍼 크기가 증가할수록 플래시 메모리 상에서 B-트리 구축 시 총비용 및 검색 시 읽기연산이 감소함을 보인다. 따라서 비교적 성능이 우수한 블록교체 기법을 사용하고 유보버퍼의 크기를 증가하여 시스템을 구현하면 플래시 메모리 상에서 대용량 데이타를 처리하는데 용이하며 장치 구성비용에 민감한 이동 컴퓨팅 장치에서도 BFTL의 노드 변환 테이블의 정보를 유지하기 위한 추가적 메모리 비용 없이 저비용으로 구현이 가능하다.

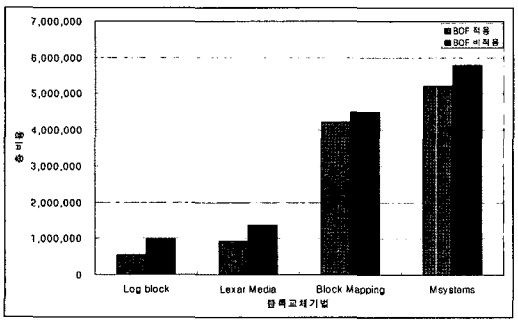

그림 11 B-트리 구축 시 수행된 총비용

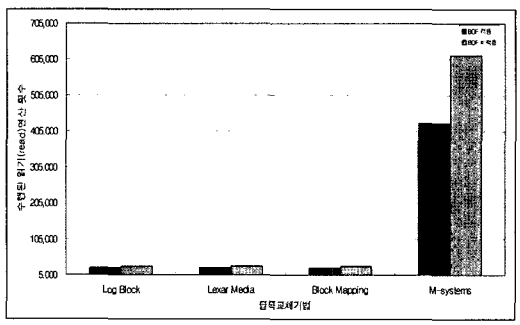

그림 12 B-트리 검색 시 수행된 읽기연산 횟수

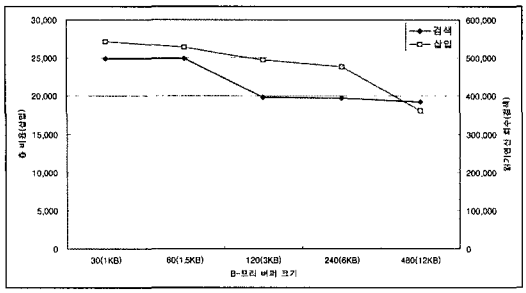

그림 13 유보버퍼 크기에 따른 총비용(삽입)/읽기연산 횟수(검색)

BOF 기법은 앞서 말한 BFTL 기법과 달리 노드 변환 테이블과 같은 추가적 하드웨어 구성비용 없이 B-트리 구축 시 비용을 줄일 수 있으며, 우수한 블록교체기법을 사용하여 덮어쓰기를 감소시키면 B-트리 구축비용도 급격히 감소한다. 하지만 BFTL 기법의 경우 플래시 메모리로 데이타를 저장할 때 항상 새로운 논리 섹터 번호를 할당하기 때문에 구축 후 유효하지 않은 데이타를 재사용하기 위한 쓰레기 수집을 블록교체기법에 상관없이 수행한다. 즉, BFTL 기법은 성능이 우수한 블록교체기법을 사용하더라도 삭제연산의 횟수는 크게 감소하지 않는 단점이 있다. 하지만 BOF 기법을 적용한 경우 기존에 플래시 메모리에 저장된 데이타에 대한 덮어쓰기를 수행하여 쓰레기 수집과 같은 추가적 작업이 필요하지 않다. 또한 비교적 성능이 우수한 블록교체기법을 적용한 BOF 기법을 사용하게 되면 B-트리 구축 및 검색비용을 상당히 줄일 수 있음을 실험을 통하여 규명하였다.

6. 결론 및 향후 계획

최근 들어, 이동 장치에 하드디스크를 대체할 매체로 플래시 메모리가 주목받고 있으며, 플래시 메모리에 저장된 데이타를 효율적으로 관리하기 위한 색인기법이 요구된다. B-트리 색인 구조는 여러 분야에서 주로 사용되는 색인기법이다. 하지만 플래시 메모리 상에서 하드디스크와 동일기법을 사용하여 효율적으로 데이타를 관리하기 위한 B-트리를 구현하면 플래시 메모리의 물리적인 특성으로 인해 심각한 성능저하를 가져올 수 있다. 따라서 본 논문에서는 플래시 메모리 상에서 효율적으로 B-트리를 설계 및 구현할 수 있는 BOF 기법을 제안하였다. BOF 기법은 플래시 메모리상의 물리적인 특성을 고려하여 유보버퍼를 사용함으로써 플래시 메모리로의 쓰기연산을 최대한 지연시켰으며, 인덱스 유닛 단위로 B-트리를 구성하여 유보버퍼의 공간적 효율성을 증가시켰다. 또한 이동 컴퓨팅 장치와 같이 장치구축비용에 비교적 민감한 장치의 특성을 고려하여 저비용으로 구현할 수 있으며, 응용프로그램 계층과 플래시 변환 계층 중간에 위치하기 때문에 기존 응용프로그램의 수정 없이 구현이 가능한 시스템 소프트웨어이다.

향후 계획으로는 플래시 메모리의 용량이 점차 증가하는 추세이고 그에 따른 대용량 데이타를 처리하기 위한 색인구조가 요구된다. 플래시 메모리 상에서 B-트리를 설계 및 구현한 본 논문에서는 B-트리 한 노드 크기를 플래시 메모리의 섹터(512B)단위로 처리하지만 하드디스크 상에서의 한 노드 크기는 블록(4KB)단위로 처리한다. 이와 같이 노드 크기가 증가하였을 때 효율적으로 플래시 메모리 상에서 노드를 관리하고 하드디스크 상의 B-트리 구축기법과 동일한 기법을 적용하여 구현할 수 있는 연구를 수행할 계획이다.

참 고 문 헌

[1] Michael Wu, and Willy Zwaenpoel, "eNVy: A Non-Volatile, Main Memory Storage System," In proceeding of 6th Symposium on Architectural Support for Programming Languages and Operating System, pp.86-87, 1994.

[2] F. Douglis, R. Caceres, F. Kaashoek K. Li, B. Marsh, and J. A. Tauber, "Storage Alternatives for Mobile Computers," In Proceedings of the 6th Symposium on Operating System Design and Implementation, pp.86-97, 1994.

[3] Intel Corporation, "Understanding the Flash Translation Layer(FTL) Specification,"http://developer.intel.com/.

[4] Intel corporation, "3 Volt Synchronous Intel StrataFlash Memory," http://www.intel.com/.

[5] 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.

[6] Amir Ban, and Ramat Hasharon, "FLASH FILE SYSTEM OPTIMIZED FOR PAGE-MODE FLASH TECHNOLOGIES," Assignee: M-systems Flash Disk Pioneers Ltd., Patent Number: 5,937,425, Date of Patent: 8/10/1999.

[7] Petro Estakhri, Berhau Iman, and Ali R. Ganjuei, "MOVING SECTORS WITHIN A BLOCK OF INFORMATION IN A FLASH MEMORY MASS STORAGE ARCHITECTURE," Assignee: Lexar Media, Inc., Patent Number: 6,145,051, Date of Patent: 11/7/2000.

[8] Takayuki Shinohara, "FLASH MEMORY CARD WITH BLOCK MEMORY ADDRESS ARRANGEMENT," Assignee: Mitsubishi Denki Kabushiki Kaisha, Patent Number: 5,905,993, Date of Patent: 5/18/1999.

[9] Chin-Hsien Wu, Li-Pin Chang, Tei-Wei Kuo, "An Efficient B-Tree Layer for Flash-Memory Storage Systems," Real-Time Computing Systems and Applications, 2003.
[10] L.P. Chang, and T.W. Kuo, "A Dynamic-Voltage-Adjustment Mechanism in Reducing the Power Consumption of Flash Memory for portable Devices," The 8th International Conference on Real-Time Computing Systems and Applications, 2002.
[11] Samsung Electronics, "256M x 8 Bit / 128M x 16 Bit NAND Flash Memory," http://www.samsung-electronics.com/.
[12] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil, "The Log-Structured Merge-Tree(LSM-Tree)," Acta Information, Vol 33, No. 4, 1996.
[13] M. Rosenblum, and J.K. Ousterhout, "The Design and Implementation of a Log-Structured File System," ACM Transaction on Computer Systems, 1992.
[14] Christophe Bobineau, Luc Bouganim, Philippe Pucheral, Patrick Valduriez, "PicoDBMS: Scaling down Database Techniques for the Smartcard," In proceeding of the 26th International Conference on Very Large Databases, Cairo, Egypt, 2000.

남 정 현

2004년 수원대학교 컴퓨터학과(학사) 2006년 숭실대학교 대학원 컴퓨터학과(석사). 2006년~현재 한화그룹 (주)한컴정보기획팀. 관심분야는 플래시메모리, 내장형 소프트웨어, 멀티미디어 데이타베이스 등

박 동 주

1995년 서울대학교 컴퓨터공학과(학사) 1997년 서울대학교 컴퓨터공학과(공학석사). 2001년 서울대학교 컴퓨터공학부(공학박사). 2001년~2003 삼성전자 책임연구원. 2004년~현재 숭실대학교 컴퓨터학부 조교수. 관심분야는 플래시메모리, 내장형 소프트웨어, 멀티미디어 데이타베이스 등