플래시 메모리 상에서 효율적인 B-트리 설계 및 구현†

남정현°, 박동주
숭실대학교 컴퓨터학부
{jhnam°, djpark}@computing.ssu.ac.kr

The Efficient Design and Implementation of The B-Tree on Flash Memory

Junghyun Nam° Dong-Joo Park
School of Computing, Soongsil University

요 약

최근 들어 PDA, 스마트카드, 휴대폰, MP3 플레이어 등과 같은 이동 컴퓨팅 장치의 데이터 저장소로 플래시 메모리를 많이 사용하고 있다. 이런 이동 컴퓨팅 장치의 데이터를 효율적으로 삽입·삭제·검색하기 위한 색인기법이 필요하다. 기존연구에서는 BFTL(B-Tree Flash Translation Layer)기법을 사용하여 플래시 메모리 상에 B-트리 구축 시 쓰기연산을 감소시켜 비용을 줄였지만, B-트리 검색비용과 하드웨어 구성비용이 증가한다는 단점을 가지고 있다. 본 논문에서는 기존 연구의 문제점을 개선하고 효율적으로 플래시 메모리상에 B-트리를 구현하기위해 BOF(B-Tree On Flash Memory)기법을 제안한다. 이 기법을 통해 BFTL 기법에 근접하는 구축비용을 얻을 수 있을 뿐만 아니라 상당한 검색비용을 줄일 수 있다. 또한 하드웨어적 비용도 고려하여 저비용으로 B-트리를 구현하였다.

1. 서 론

최근 들어 PDA, 스마트카드, 휴대폰, MP3 플레이어 등과 같은 이동 컴퓨팅 장치의 데이터 저장소로 플래시 메모리를 많이 사용하고 있다.

플래시 메모리는 전력 소모가 적고 충격에 강하며 전원이 꺼지더라도 저장된 데이터가 사라지지 않는 비휘발성 메모리 특성을 가지고 있다. 플래시 메모리는 다수 개의 블록(block)으로 구성되며 하나의 블록에는 32개의 섹터(sector)가 포함되며 하나의 섹터는 512 바이트의 데이터를 저장하는 영역(data area)과 16바이트의 데이터를 저장하는 영역(spare area)으로 구성된다. 하드디스크와 달리 데이터가 저장된 섹터에 대해 중첩쓰기(overwrite)가 허용되지 않기 때문에 플래시 메모리에 데이터를 쓰기 위해서는 삭제(erase)연산이 선행되어야 하는데 삭제연산은 블록단위로 수행되며 읽기·쓰기단위인 섹터단위 보다 수행시간이 오래 걸린다[1]. 이러한 플래시 메모리의 단점을 극복하고자 플래시 변환 계층(Flash Memory Translation Layer)이라는 시스템 소프트웨어를 사용한다. 플래시 변환 계층은 플래시 메모리의 삭제연산을 감추기 위하여 플래시 메모리와 호스트 시스템의 파일시스템 사이에 위치한다. 삭제연산은 쓰기연산이 수행될 때 파일시스템이 생성한 논리 섹터 주소(Logical Sector Address)를 플래시 메모리상에 이미 삭제연산을 수행한 빈 블록의 물리주소로 재 사상(mapping)함으로써 감춰진다. 즉, 하드디스크와 같은 단일 저장 장치를 구성함으로써, 일반 파일시스템을 사용하여 플래시 메모리를 효율적으로 제어할 수 있다.

플래시 변환 계층의 기법들은 여러 블록교체기법을 사용한다. [3,4,5]에서의 블록교체기법들은 한 섹터에 대한 중첩쓰기연산이 발생하게 되면 미리 할당된 여유블록(spare block)에 수정할 데이터를 할당하고 이를 연결리스트로 구성해 읽기연산을 수행할 때 역순으로 검색하여 가장 최신의 데이터를 제공하는 방법이다. 이처럼 플래시 변환 계층은 주소 변환 테이블을 이용하여 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. 이 같은 플래시 변환 계층의 한계 때문에 직접적으로 플래시 메모리상에 B-트리를 구축하는 것은 많은 비용을 요구하게 된다.

플래시 메모리 상에 B-트리를 구현하기 위해 [6]에서 BFTL 기법을 제안하였으며 본 논문에서는 BFTL 기법의 단점을 극복하고 플래시 메모리상에서 B-트리를 보다 효율적으로 구현하기 위해 BOF(B-tree On Flash Memory) 기법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존에 연구된 기법과 본 논문에서 제안하는 BOF 기법을 기술하고 3장에서는 두 기법 간의 성능평가를 기술하며 마지막으로 4장에서 결론을 맺는다.

2. B-트리 구현 기법

B-트리 구조는 데이터의 삽입·삭제·검색을 효율적으로 처리하기위해 가장 널리 사용되는 색인구조 중 하나이다. 플래시 메모리상에 B-트리를 구축할 때 중요한 문제는 쓰기연산을 줄이는 것이다.

본 장에서는 플래시 메모리상에서 B-트리를 구축하기위해 기존에 연구된 BFTL 기법에 대해 기술하고 BFTL 기법의 단점을 극복하여 효율적이고 경제적인 B-트리를 구현하기위해 본 논문에서 제안하는 BOF 기법에 대해 기술한다.

2.1 BFTL(B-Tree Flash Translation Layer)

플래시 메모리 상에 B-트리를 구축할 때 빈번한 읽기·쓰기연산의 발생은 플래시 메모리의 성능저하를 유발시킨다. 그림 1은 BFTL 기법에서 사용하는 B-트리 노드구성 단위인 색인유닛(index unit)을 나타낸 것으로 노드안의 각 엔트리(entry)정보를 유지하는 역할을 한다. 그림 2는

†본 연구는 숭실대학교 교내연구비 지원으로 이루어졌음.

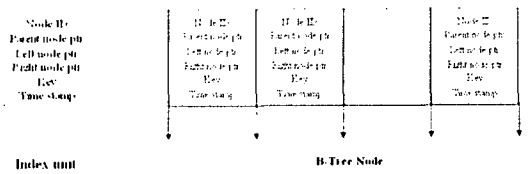

그림 1 BFTL 색인유닛(Index unit)

BFTL 기법의 구성 및 구현방법을 도식화한 것이다. BFTL 기법은 새로 입력된 색인유닛을 임시로 저장하는 유보 버퍼(Reservation Buffer), 플래시 메모리에 저장된 색인유닛의 논리 섹터 주소를 기록한 노드 변환 테이블(Node Translation Table), 유보버퍼 내에 있는 색인유닛을 플래시 메모리로 저장하기 위한 수용정책(Commit Policy)으로 구성된다. B-트리와 관련된 응용프로그램이 새로운 키 값을 입력하면 유보 버퍼와 노드 변환 테이블의 해당 노드 리스트를 이용하여 키 값이 삽입될 단말 노드(leaf node)를 구성하고 키 값을 해당 노드에 삽입하여 색인유닛을 생성한다. 생성된 색인유닛은 일시적으로 유보 버퍼에 저장되며 유보 버퍼 내에 빈 공간이 없을 경우 수용정책에 의해 쓰기연산이 수행되어 플래시 메모리에 저장된

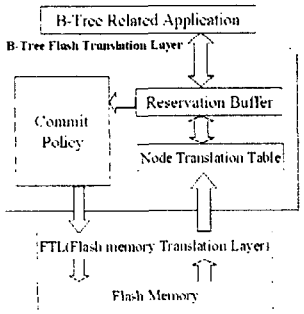

그림 2 BFTL 구성 및 구현

다. BFTL 기법은 앞서 말한바와 같이 색인유닛으로 노드를 구성하기 때문에 각 색인유닛이 수용정책에 의해 플래시 메모리로 저장될 때 유보버퍼 내에 서로 다른 노드에 속하는 색인유닛들이 플래시 메모리상의 동일한 섹터에

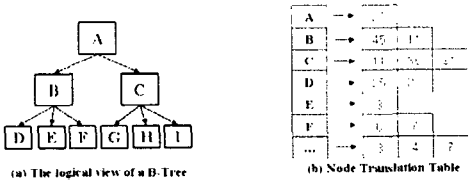

그림 3 노드 변환 테이블

저장될 수 있다. 따라서 플래시 메모리상에 흩어진 색인유닛들을 관리하기 위하여 RAM과 같은 휘발성 메모리에 그림 3과 같이 노드 변환 테이블을 추가하여 색인유닛의 저장된 논리 섹터 주소를 리스트로 유지한다. 예를 들어 "C" 노드에 해당하는 색인유닛은 플래시 메모리의 논리 섹터 주소 11, 36, 21에 저장되어 있다. BFTL 기법은 B-트리를 구축 시 유보버퍼를 사용함으로써 쓰기연산을 감소시켰지만 노드구성 과 검색 시 노드 변환 테이블의 리스트를 참조하기 때문에 읽기연산이 많이 발생하여 검색 비용이 증가하게 된다. 뿐만 아니라, 유보버퍼와 노드 변환 테이블을 추가하기 위한 하드웨어 비용이 증가 한다는 문제를 갖는다. 이러한 문제점을 극복하기 위해서 BOF 기법을 제안한다.

2.2 BOF(B-Tree On Flash Memory)

BOF 기법은 BFTL 기법과 동일한 색인유닛을 사용하여 버퍼의 활용성을 높였으며 검색비용을 줄이기 위해서 같은 노드에 속하는 유보버퍼내의 색인유닛들을 플래시 메모리상의 동일한 섹터에 저장한다. 한 섹터에 한 노드를 저장함으로써 특정 노드를 구성할 때 한 번의 읽기연산을 수행하여 노드를 구성할 수 있으며, 노드변환 테이블을 유지하기 위한 추가적 하드웨어 비용이 필요 없다. 따라서 이동 컴퓨팅 장치와 같은 하드웨어 비용한계가 있는 매체에서도 저비용으로 구현이 가능하다. 그림 4는 BOF 기법의 구성과 버퍼관리를 도식화한 것이다. 입력된 키

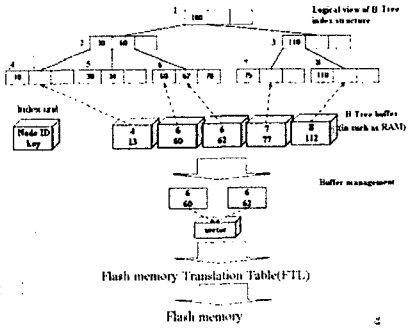

그림 5 BOF 구성 및 버퍼 관리

값을 해당 노드에 삽입하기 위해선 삽입될 단말 노드를 검색한 후 플래시 메모리에 저장된 해당 노드 데이터와 B-트리 버퍼안의 색인유닛과의 노드 병합(node merge)을 수행한다. 마지막으로 병합된 노드에 새로 입력된 키를 삽입함으로써 B-트리 노드를 구성하게 된다. 그림 5와 같이 "4" 번 노드에 새로운 키 값이 삽입된다면 플래시 메모리상의 "4" 번 노드를 읽어 B-트리 버퍼에 상주하고 있는 최신의 색인유닛과 병합한다. 만약 병합된 노드에 빈 공간이 없다면 노드분리(node split)가 발생하는데 노드분리가 발생하면 버퍼관리에 의해 새로 생성된 노드와 기존의 노드를 플래시 메모리로 저장한다. 노드분리 후

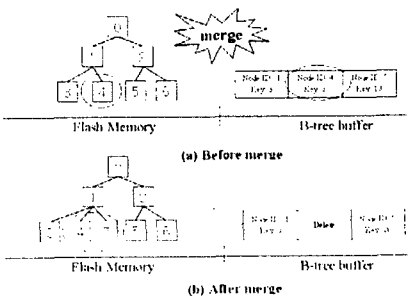

그림 5 노드 합병

B-트리 버퍼에 삽입하지 않고 플래시 메모리에 직접 저장함으로써 노드분리가 발생할 때 기존 노드의 쓰레기 값을

제거한다. 버퍼 관리는 노드가 분리할 때 수행하기도 하지만 B-트리 버퍼에 빈 공간이 없으면 B-트리 버퍼의 첫 번째 색인유닛의 노드 아이디(node ID)와 동일한 색인유닛들을 함께 플래시 메모리로 저장하는 방법을 사용한다. 위와 같이 BOF 기법은 B-트리를 구축할 때 자주 참조되는 데이터를 B-트리 버퍼에 저장하여 쓰기연산을 줄일 수 있으며, 플래시 메모리상으로의 쓰기연산이 발생할 경우 효율적인 블록교체기법[3,4]을 사용함으로써 구축비용을 줄일 수 있다.

3. 성능 평가

본 장에서는 실험을 통한 성능분석을 통하여 본 논문에서 제안하는 BOF 기법의 우수함을 규명한다.

B-트리 구축에 관한 실험에서 한 노드가 최대 가질 수 있는 엔트리의 수를 7, B-트리 버퍼의 사이즈를 30으로 정하고 랜덤 데이터 10,000개를 이용하여 B-트리를 구축하였다. B-트리를 구축할 때 플래시 메모리 내의 읽기·쓰기·삭제연산에 따른 횟수를 구하고 전체 연산비용을 계산하였다. B-트리 검색실험은 임의의 데이터 5,000개를 이용하여 읽기연산의 횟수를 계산한다.

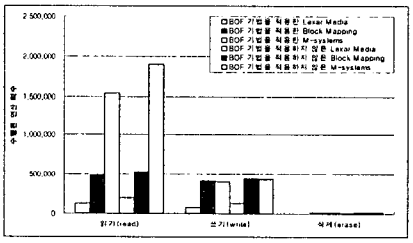

그림 6 랜덤 데이터에 대한 B-트리 구축 시 연산비용

그림 6은 랜덤 데이터를 입력하여 B-트리를 구축할 때의 연산횟수를 나타내며 그림 7은 플래시 메모리의 각 연산에 따른 상대비용(읽기 : 1, 쓰기 : 7, 삭제 : 63)을 적용한 결과이다. 그림 6에서 랜덤 데이터가 입력되면 BOF 기법을 적용한 블록교체 기법의 경우 B-트리 버퍼를 사용함으로써 각 연산비용이 감소함을 알 수 있다. 그림 7에서 BOF 기법을 적용한 블록교체기법의 총 비용이 감소

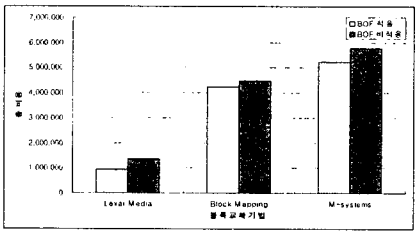

그림 7 랜덤 데이터에 대한 B-트리 구축 시 총 비용

한다. 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 중첩쓰기연산을 감소시키면 B-트리 구축비용도 급격히 감소한다. 하지만 BFTL 기법의 경우 플래시 메모리로 데이터를 저장할 때 항상 새로운 논리 섹터 주소를 할당하기 때문에 구축 후 유효하지 않은 데이터를 제거하기 위한 쓰레기 수집을 블록교체기법에 상관없이 수행한다. 즉, BFTL 기법은 성능이 우수한 블록교체기법을 사용하더라도 삭제연산의 횟수는 크게 감소하지 않는 단점을 가지고 있다. 그림 8은 비교적 성능이 우수한 Lexar Media[4]

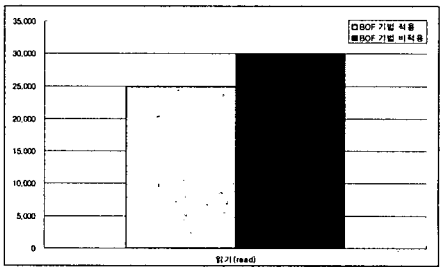

그림 8 검색비용 결과

블록교체기법을 BOF 기법에 적용하여 검색을 수행했을 때의 비용을 나타낸다. 그림 8과같이 BOF 기법에 Lexar Media[4] 블록교체기법을 적용하여 검색할 때 비용이 적게 요구되는 것을 알 수 있다. 그림 7,8에서 우수한 블록교체기법을 적용한 BOF 기법을 사용하게 되면 B-트리 구축 및 검색비용을 상당히 줄일 수 있음을 실험을 통하여 규명하였다.

4. 결론

본 논문에서는 플래시 메모리상에 B-트리를 구현하기위한 BOF 기법을 설계하고 성능을 분석하여 그 우수성을 규명하였다.

플래시 메모리상에 B-트리를 구현하기 위해서 고려해야 할 문제로는 구축할 때 쓰기연산의 감소, 검색할 때 읽기연산의 감소, 하드웨어적 비용절감을 들 수 있다. 본 논문에서 제안한 BOF 기법은 위 고려사항을 최대한 반영하며 다양한 환경에 적용 가능하기 때문에 플래시 메모리상에 B-트리를 구현하기위한 최적의 기법이다.

<참고문헌>

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

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

[3] Amir Ban, 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.

[4] Petro Estakhri, Berhau Iman, 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.

[5] 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.

[6] Chin-Hsien Wu, Li-Pin Chang, Tei-Wei Kuo, "An Efficient B-Tree Layer for Flash-Memory Storage Systems", RTCSA, 2003.