F-Tree : 휴대용 정보기기를 위한 플래시 메모리 기반 색인 기법

변 시 우*

F-Tree : Flash Memory based Indexing Scheme for Portable Information Devices

Siwoo Byun*

Abstract

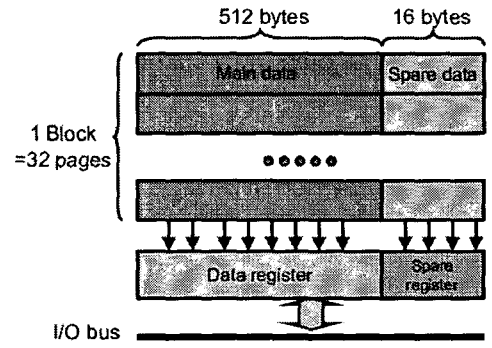
Recently, flash memories are one of best media to support portable computer's storages in mobile computing environment. The features of non-volatility, low power consumption, and fast access time for read operations are sufficient grounds to support flash memory as major database storage components of portable computers. However, we need to improve traditional Indexing scheme such as B-Tree due to the relatively slow characteristics of flash operation as compared to RAM memory. In order to achieve this goal, we devise a new indexing scheme called F-Tree. F-Tree improves tree operation performance by compressing pointers and keys in tree nodes and rewriting the nodes without a slow erase operation in node insert/delete processes. Based on the results of the performance evaluation, we conclude that F-Tree indexing scheme outperforms the traditional indexing scheme.

Keywords : Tree Indexing, Portable Devices, Dlash Memory, Database, Simulation

논문접수일 : 2006년 08월 14일 논문게재확정일 : 2006년 12월 11일
※ 이 논문은 2005년도 학술진흥재단의 지원을 받아 수행된 연구입니다(KRF-2005-041-D00653).
* 안양대학교, 부교수, (430-714) 경기도 안양시 만안 안양5동 안양대학교 디지털미디어학부, TEL : 031-467-0922,
E-mail : swbyun@anyang.ac.kr

1. 서 론

최근 휴대형 소형 정보기기들의 정보 저장용 미디어로 플래시 메모리가 많이 사용되고 있다. 플래시 메모리는 영구 저장이 가능하다는 측면에서 하드 디스크와 유사하지만, 내충격성, 휴대성, 접근속도, 무진동성, 무소음성, 부품크기 측면에서 비교할 수 없을 정도로 매우 우수하다. 전력 소모량도 10㎃ 정도인데, 이는 저 전력 메인 메모리의 8분의 1정도에 불과하다. 또한, 메인 메모리와 비교하면, 메인 메모리의 최대 약점인 영구저장 불가능성(휘발성)을 해결했다는 측면에서 더 우수하므로, 차세대 기억장치라고 한다. 대부분의 소형 정보기기에서는 공간제약, 전력소모, 중량 및 내충격성 문제로 하드 디스크는 사용할 수 없기 때문에, 비휘발성 저장장치로서 플래시 메모리를 반드시 사용하여야만 한다. 다만, 일반 메인 메모리와는 달리, 쓰기와 소거 연산에 상당히 많은 시간이 소요되며, 쓰기 횟수가 최대 100,000에서 1,000,000번 정도로 제한되는 고유한 특성(약점)이 있다[Keun Soo Yim, 2003].



〈그림 1〉 NAND 플래시 메모리구조

플래시 메모리는 일반 메인 메모리와 달리 바로 쓰기 연산을 수행할 수 없고, 이전에 미리 블록 단위로 포맷 개념의 소거 연산을 수행한 후 쓰기 연산을 수행할 수 있다. 읽기 및 쓰기 연산도 메인 메모리처럼 워드나 바이트 단위가 아니고, <그림 1>의 예[Chanik Park, 2004]와 같이 페이지 단위로 수행된다. 즉, 한 블록은 32개의 페이지로 구성되어 있으며, 한 페이지는 512바이트의 메인 데이터와 16바이트의 보조 데이터로 구성된다. 보조 데이터는 불량 블록 정보나 에러 정정 코드 정보 등을 저장한다. 쓰기 연산과 읽기 연산은 크기가 작은 페이지 단위(512B)로 수행되지만, 소거 연산은 크기가 큰 블록 단위(16KB)로 수행된다.

이러한 플래시 메모리 기반의 데이터 저장 장치를 통하여 신속하게 저장하고 효율적으로 검색을 위해서는 플래시 메모리의 특성을 고려한 효율적인 색인 구조와 저장 기법이 필요하다. 그 이유는 플래시 메모리에는 기존의 하드 디스크나 메인 메모리와는 전혀 다른 다음의 특성(약점)들이 존재하기 때문이다.

첫째, 읽기 연산의 경우에는 플래시 메모리의 연산 처리 속도가 하드 디스크에 비하여 800배 정도로 매우 빠르며, 일반 RAM 메모리에 비해서는 좀 느리지만 10㎲ 정도로 빠르므로 접근시에 별 문제가 없다[이옥희, 2004]. 하지만, 플래시 메모리의 쓰기 연산의 경우에는 속도가 읽기 연산 대비 20배 정도의 많은 시간이 소모되어 매우 느리며, 메인 메모리처럼 Update-In-Place(제자리 덮어 쓰기)가 불가능한 Update-Out-Place 구조이다.[Samsung, 2006] 더욱이 쓰기 연산 전에 2㎳정도의 느린 속도로 소거 연산을 반드시 수행해야 하므로 연산의 총 부담이 크게 늘어난다(<표 1> 참조).

둘째, 하드 디스크나 메인 메모리는 쓰기 횟수가 거의 제한이 없는 반영구적 수명을 가지고

있는 반면에, 플래시 메모리는 쓰기 횟수가 최대 1,000,000번 정도로 수명이 제한된다. 즉, 이 수명을 넘기면 더 이상 쓰기 연산을 수행할 수 없다.

〈표 1〉 다양한 메모리 소자의 특성 비교

Storage Media	Read Operation	Write Operation	Erase Operation
RAM (Main Memory)	2.6㎲(512B)	2.6㎲(512B)	-
NOR-Type Flash Memory	15㎲(512B)	3.5㎳(512B)	1.2s(128KB)
NAND-Type Flash Memory	36㎲(512B)	226㎲(512B)	2㎳(16KB)
Hard Disk	12.4㎳(512B)	12.4㎳(512B)	-

그러나 아직 플래시 미디어 기반의 색인 기법에 대한 연구는 시작 단계이다. 이유는 그 동안 플래시 메모리는 주 메모리의 성능과 비교하면 느리면서도 고가이며, 일반적인 시스템에는 사용되지 않는 특수 부품이며, 디스크에 비하여 저장 비용이 고가였기 때문이다. 이러한 이유로 일부 시스템 환경 설정치 정도만을 저장하는 용도로 주로 사용되어 많은 데이터를 저장하고 색인까지 관리할 필요가 과거에는 없었기 때문이다.

그러나 최근 플래시 메모리 기술의 발전으로 대부분의 단점들이 해소되었고, 휴대용 정보기기에도 점점 더 많은 데이터를 저장하게 되었다. 또한, 일반 PC에 비하여 휴대형 소형 정보기기의 빈약한 자원(CPU, RAM) 조건을 가만할 때, 플래시 메모리 데이터의 색인 구조 및 접근 방식은 시스템 성능에 상당히 큰 영향을 미친다고 한다[Chin-Hsien, 2003a]. 따라서 플래시 메모리는 저장 신뢰성과 성능 측면에서, 기존의 디스크나 RAM에서는 전혀 존재하지 않았던 이러한 고유한 특성을 고려한 색인 저장 기술을 필요로 한다. 특히, 향후 플래시 메모리는 매년 2배의 용량 증가와 가격하락으로 인하여 현재의 PC나 노트북의 하드 디스크를 대체할 가능성이 매우 크다고 한다[Li-Pin Chang, 2004]. 따라서 이러한 향후 수요에 대비하여 효과적인 저장 기법과 색인 기술에 대한 연구가 요구된다.

2. 관련 연구

현재의 데이터 저장 시스템에서 사용되는 일반적인 색인 기법에 대한 기존의 연구를 분류해 보면, 크게 디스크 기반 색인 시스템과 메인 메모리 기반 색인 시스템으로 접근 방향을 나눌 수 있다.

개념적으로 디스크 기반 색인 및 저장 기술의 목표는 디스크의 접근 횟수와 디스크 공간을 최소화하는 것이며, 디스크 I/O를 가장 큰 비용으로 고려한다. 그러나 메모리 기반 색인 및 저장 기술은 디스크의 접근이 없으므로, CPU 수행 시간을 줄이고, 최소한의 메모리 공간을 사용하는 것이 중요하다. 저렴하고 대용량인 디스크 기반 저장 방식이 저장 비용 측면에서는 유리하지만, 아무리 I/O 버퍼를 많이 할당하더라도 속도 측면에서는 메모리 기반 저장 방식보다는 매우 느리다. 그러나 두 방식 모두 플래시 메모리 기반 저장 환경에는 부적합하므로 플래시 메모리의 고유한 특성을 고려하여 새로 개발하여야 한다.

(1) 디스크 기반 색인 시스템

디스크 기반 색인 및 저장 시스템에서는 검색 시에 디스크 접근을 최소화하기 위하여 노드의 크기를 디스크 페이지와 같은 크기나 배수로 설정하고, 되도록이면 많은 엔트리를 한 노드에 넣어야 유리하다. 한 노드에 많은 엔트리가 들

어갈 경우 모든 엔트리를 검색해야 하기 때문에 검색 시에 연산 처리 성능은 당연히 저하된다. 그러나 디스크 기반 저장 시스템에서는 이러한 검색 성능 저하보다는 디스크 접근에 의한 성능 저하가 훨씬 더 크기 때문에 한 노드에 많은 엔트리를 넣는 방향으로 설계한다[Cha, 1997]. 주로 B-Tree, B^+-Tree[황규영, 2000] 계열의 인덱스가 많이 사용되며, 공간 색인으로는 R-Tree, R^*-Tree[Beckmann, 1990]계열이 사용되고 있다.

R-Tree계열의 인덱스는 일반적으로 디스크 기반 색인으로서 메인 메모리를 사용하지 않는 공간 인덱스로 구현되었다. R-Tree는 삽입 연산, 삭제 연산, 분할이나 병합과 같은 리벨런싱 연산이 수행될 경우, 동일한 위치로 많은 섹터가 판독 또는 재기록 되는 부담이 있다. 디스크 기반 시스템에서는 이러한 연산들의 검색 효율을 위하여 디스크의 연속 섹터에 그룹핑되어 있으며, 디스크 특성을 잘 고려한 R-Tree는 디스크 기반 시스템에서 실제로 매우 효과적이다. 그러나 최근에 많이 연구되고 있는 GPS 기반 이동체나 휴대폰의 시공간 색인의 경우에는 이동체의 수가 많거나 위치변화가 많으면 실시간 처리를 어렵게 만드는 디스크 병목현상이 자주 발생한다. 이를 해결하기 위하여 디스크 기반의 한계를 극복한 메인 메모리 색인이 필요하며, 디스크와 메인 메모리를 합친 통합 색인 기법도 제안되었다[이창우, 2003].

(2) 메인 메모리 기반 색인 시스템

메인 메모리 기반 색인 및 저장 시스템은 디스크 기반 시스템에 비하여 디스크에 접근하는 시간이 대폭 줄어들기 때문에 훨씬 더 빠른 성능을 보여줄 수 있다. 그러나 문제는 시스템 전원이 차단되는 등의 치명적인 장애가 발생할 경우이다. 디스크 기반 저장 시스템은 디스크에 데이터를 일일이 저장하므로 장애에 매우 강하다. 반면에 메인 메모리 데이터베이스는 메모리에 저장된 데이터가 사라지므로 디스크 기반 저장 시스템과는 다른 백업, 복구 방식이 필요하다.

메인 메모리 기반 색인 시스템에서는 일반적으로 T-Tree가 1차원 데이터를 위한 색인으로서 좋은 성능을 보이며, 비교적 적합하다고 알려져 있다[Lehman, 1986]. T-Tree는 이진 검색과 높이 균형을 가지고 O(logN)의 트리 순회가 가능한 AVL-Tree의 빠른 검색 특성을 가지고 있으며, 한 노드 안에 여러 개의 데이터를 가지고 저장효율이 좋은 B-Tree의 성질도 함께 가지고 있다. T-Tree는 빠른 처리속도와 메모리 사용의 최적화라는 메인 메모리의 특성에 적합한 구조로 알려져 있다[이창우, 2003]. 그러나 [Hongjun, 2000]에서 T-Tree는 동시성 제어에 대한 고려가 매우 부족하였으며, 이를 고려한다면 B-Tree가 성능을 추월할 수 있음을 밝혔다. 또한 실제로는 T-Tree와 함께 성능 향상을 위하여 노드에 자료 구조를 개선한 B-Tree 계열의 개선된 인덱스가 많이 사용된다.

디스크 기반 색인은 노드의 접근 횟수가 디스크의 I/O의 수와 같으므로, 트리의 깊이는 성능에 큰 영향을 미쳤다. 삽입/검색 시에는 데이터를 삽입/검색할 노드를 찾기 위하여 비교하면서 내려가는 노드의 개수가 성능에 가장 큰 영향을 준다. 따라서 깊이가 얕고 넓게 퍼진 트리를 써서 삽입/검색 시에 I/O 비용을 최소화 하였다. 반면에, 메모리 기반 색인의 접근 비용은 포인터로 노드의 메모리 주소를 획득하는 비용이므로 크지 않다. 따라서 디스크 기반 색인에서 선호하는 얕고 넓게 퍼진 트리 구조는 더 이상 유용하지 않다. 메모리 기반 색인에서는 디스크 기반 색인과는 달리 노드의 용량을 변화시켜 트리의 깊이와 비교횟수를 조절하여 성능 향상이 가능하다[이창우, 2003].

그러나 플래시 메모리 기반 저장 시스템에서

는 기존의 인덱스 구조와 관리 기법을 그대로 적용할 수 없다. 이유는 다음과 같다.

- 플래시 메모리는 부품 특성상 단 한 바이트라도 쓰기 연산을 수행하기 위해서는 이전에 세그먼트 단위의 매우 느린 소거 연산이 먼저 수행되어야 한다. 또한, 기록과 소거 연산은 판독 연산에 비하여 각각 20배, 200배 정도로 매우 느리다[Samsung, 2006]. 그러므로 쓰기 연산을 판독 연산과 같은 비중으로 적용할 수 없다.
- 쓰기 횟수가 최대 1,000,000번 정도로 제한되어 있으며, 특정 한 바이트가 제한 횟수를 초과하면 그 바이트를 포함한 그 세그먼트의 블록 전체를 사용할 수 없으며, 이러한 불량 세그먼트는 곧 시스템 운영상에 치명적인 장애 요인이 된다.
- 쓰기 연산의 전력소모량은 읽기 연산의 9배 정도가 되므로[Lee, 2003] 배터리 용량이 매우 제한적인 휴대용 정보기기에 9배 정도의 큰 전력 부담을 준다.

이와 같은 세 가지 이유로 쓰기 연산의 횟수를 최대한 줄여야만 성능 저하를 막고 플래시 메모리의 안정성과 수명도 증대시킬 수 있다.

3. 플래시 메모리의 특성을 고려한 색인 저장 및 관리 기법 제안

3.1 제안 배경

본 논문에서는 디스크 기반 및 메인 메모리 기반 색인 중에서 가장 보편적인 B-Tree 색인을 근간으로 하여 플래시 메모리에 적합한 색인 저장 기법을 연구하였다. 메인 메모리 데이터베이스에서 많이 사용되는 T-Tree도 동시성 제어를 고려할 경우 B-Tree 보다 성능이 낮으며, 그 이유는 메모리의 발전 속도 보다 CPU의 발전 속도가 더 빠르므로, 상대적으로 메모리 접근수가 적은 B-Tree가 유리하기 때문이다[Hongjun, 2000]. 또한, T-Tree도 B-Tree에서 파생되어 B-Tree의 속성을 가지고 있으며, 실제 메모리 데이터베이스에서 B-Tree도 많이 사용된다. 또한, 디스크 기반 데이터베이스에서도 B-Tree가 대부분 활용된다는 관점에서 본 논문에서는 B-Tree에 기초하였고, B-Tree 중에서도 더 진보되어 보편적인 B^+-Tree를 대상으로 하였다. 그 이유는 B^+-Tree는 B-Tree와는 달리 중간 노드에 데이터 관련 정보를 넣지 않고 리프 노드에서 저장하므로, 상대적으로 색인 자체의 저장 효율이 높아지기 때문이다. 또한, 우선적으로는 B^+-Tree에 적용하였지만, 제안 기법을 B-Tree를 포함한 일반적인 트리 구조의 색인에도 적용이 가능하다.

이러한 관점에서 본 연구에서는 메모리 기반 데이터베이스와 디스크 기반 데이터베이스에서 근간이 되는 B-Tree를 플래시 메모리 데이터베이스에 적합하게 개선하여, 쓰기 연산과 지우기 연산의 부담을 줄이고, 성능을 개선할 수 있는 새로운 색인 저장 기법을 제안하고자 한다.

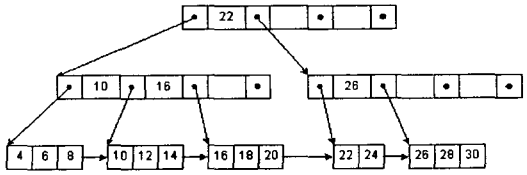
플래시 메모리 기반 저장장치의 인덱스 연산의 성능 향상을 위한 방법으로 *B-Tree Flash Translation Layer*(BFTL) [Chin-Hsien, 2003b]이 제안되었다. BFTL은 기본적인 B-Tree상에서 인덱스 유닛의 예약 버퍼링을 통한 노드 재정렬을 통하여 중복된 쓰기 연산의 수를 최적화시키는 기법이다. 반면, BFTL은 노드 구성과 검색시 노드 변환 테이블 참조를 위한 읽기 연산의 증가와 예약 버퍼와 노드 변환테이블의 하드웨어 구현 비용 증가의 오버헤드가 수반된다. 이 기법과는 달리, 본 연구는 B-Tree의 상용화된 형태인 B+-Tree를 대상으로 하여 인덱스 노드를 압축하고 재기록 함으로써 인덱스 노드의

저장 횟수를 증가시키는 재활용에 포커스를 두고 있다. 또한, 플래시 메모리만의 고유한 물성적인 일방식 저장 특성을 이용하여서 성능개선의 측면이 다르다. 따라서 본 제안기법의 상위층에 BFTL의 기법이 독립적으로 존재하며, 만일 공존하게 되면 성능상의 시너지 효과를 낼 수 있다. 즉, BFTL에서 내려 보내는 최종적인 인덱스 유닛을 본 제안 기법이 받아서 다시 처리하면 저장 효율성을 추가로 더 향상시킬 수 있다. 즉, 본 제안 기법은 BFTL이 위에 있다고 가정하며, BFTL 바로 아래에 위치하는 모듈이며, 임의의 FTL을 기반으로 하여 높은 성능이 나오도록 연동가능하다.

3.2 플래시 메모리 기반 색인 저장 기법의 제안

B-Tree 계열의 색인은 데이터의 삽입, 삭제, 검색을 효율적으로 처리하기 위하여 가장 널리 사용되는 색인 구조이다. B-Tree의 삽입, 삭제, 리벨런싱은 많은 노드들이 읽혀지고 동일한 위치에 다시 쓰여지게 한다. 그러나 플래시 메모리 상에 B-Tree를 그대로 구현하게 되면, 느린 쓰기 연산을 통하여 입출력이 이루어지므로 성능이 크게 저하되며, 동일한 위치에 반복 기록되어 안정성도 저하되게 된다[Chin-Hsien, 2003a]. 따라서 플래시 메모리 상에 B-Tree 계열의 색인 구축할 때 중요한 문제는 쓰기 연산을 최대한 줄이는 것이다[JungHyun Nam, 2005].

기본적으로 B-Tree는 하위 노드에 대한 포인터 정보와 함께 데이터 관련 정보를 한 노드 안에 보관한다. 그러나 아래 그림2에서 보는 바와 같이 B^+-Tree는 중간 노드에 데이터 관련 정보를 넣지 않고 리프 노드에 이를 저장한다. 즉, 중간 노드에서는 순수한 색인 정보만을 저장하므로, 색인 자체의 저장 효율이 높아진다. 또한, 리프 노드에서는 색인 검색의 도움 없이 바로 순차적인 접근이 가능한 장점도 있다. 따라서 플래시 메모리의 색인으로서는 기본적인 B-Tree보다도 B^+-Tree가 더 적합하다.



〈그림 2〉 B+-Tree 색인의 예

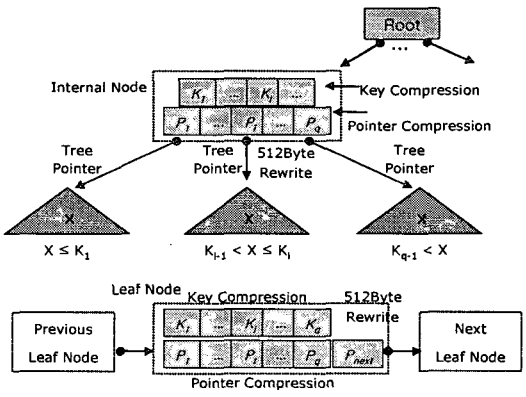
그리고 B^+-Tree에서 수많은 임의의 값들을 삽입하고 삭제하는 시뮬레이션을 수행하여 분석한 결과 69%정도 차 있을 때가 가장 효율적인 것으로 나타났다. 한 노드에 최대한 많이 저장하면 검색 노드수도 줄어들고 저장 효율은 향상되지만, 삽입, 삭제시 노드의 변동이 너무 빈번하여 많은 수의 쓰기 연산을 유도하여 결과적으로 더 손실이 크다. 이러한 분석 결과와 기타 B^+-Tree에 대한 자세한 이론은 [황규영, 2000]에 구체적으로 잘 설명되어 있다.

이러한 B^+-Tree를 플래시 메모리상에 구현하면, 트리 노드의 삽입 및 삭제로 인한 쓰기 연산이 빈번히 발생하는데, 이를 줄이는 것이 중요하다. 전술한 바와 같이 쓰기 연산은 느린 소거 연산을 수반하여 성능을 저하시키고, 플래시 메모리의 수명을 단축시키기 때문이다. 본 연구에서는 플래시 메모리의 쓰기 연산과 소거 연산의 세부적인 특성을 이용한 이어쓰기 기법과 노드 압축 기법을 활용하였다.

플래시 메모리는 소거 연산후, 쓰기 연산을 수행할 때 각 비트를 0에서 1로 변환하는 것은 불가능하다. 하지만 1에서 0으로 쓰기 연산을 수행하는 것은 가능하다. 만일 0에서 1로 변환하고자 한다면, 다시 모든 비트를 1로 초기화하

는 소거 연산을 먼저 수행하여야 한다. 그러나 1에서 0으로의 쓰기 연산은 소거 연산의 수행 없이도 가능한 특수한 성질이 있다[정재용, 2002]. 이 특성을 잘 활용하면 높은 성능 개선효과를 낼 수 있다. 즉, 한 페이지에 1에서 0으로 쓰기는 소거 연산 없이도 반복해서 수행이 가능하다. 또한, 소거 연산 부담이 제거되므로, 플래시 메모리의 수명이 그 만큼 연장되고 저장 시간도 단축된다. 따라서 한 페이지에 쓰고자 하는 내용을 압축해서 앞부분부터 저장하고, 필요시 남은 뒷공간에 다시 저장이 가능하다. 이 기능은 쓰기가 한번만 가능한 보통의 CD-ROM에서, 저장할 용량이 작을 경우 먼저 일부 쓰기를 수행한 후, 나중에 추가 데이터가 발생하면, 나머지 빈 공간에 다시 이어쓰기를 하는 것과 유사하다.

실제로 플래시 메모리에서 블록내의 쓰기 연산이 무한히 반복 가능하지는 않으며 제한된 쓰기 횟수(NOP)를 초과할 수 없다. 만일 제한된 쓰기 횟수를 초과하게 되면, 상당히 느린 소거 연산을 다시 실행하여야만 해당 블록의 쓰기가 다시 가능하여 진다. 그러나 본 논문에서 한 블록에 대하여 쓰기 횟수는 다시 쓰기를 포함하여 2회이므로 기본적으로는 제안 기법이 적용가능



〈그림 3〉 F-Tree 의 중간 노드와 리프 노드의 구조〉

하다. 따라서 블록 쓰기에 대한 NOP이 최소 2 이상으로 보고 본 기법을 적용하였다. 즉, NOP이 1인 제한된 경우는 적용이 불가능하다.

본 연구에서 이러한 노드에 대한 압축과 이어쓰기의 복합 연산을 간단히 "압축이어쓰기"라고 하고, 이러한 특성을 B^+-Tree에 반영하여 개선한 구조를 "F-Tree"라고 하였다. <그림 3>은 제안된 F-Tree의 중간 노드와 리프 노드의 구조이다.

노드 압축은 일반적인 압축 알고리즘을 적용하되, 압축 효율을 높이기 위하여, 키 부분과 포인터 부분을 분리하여 압축한다. 다음절의 실험에서 설명하겠지만, 이러한 간단한 노드 압축만으로도 해당 노드에 쓰기 횟수를 반으로 줄일 수 있었다. 만일 키와 포인터에 대한 특수한 압축 기법을 적용하면, 복잡하기는 하지만 더 큰 압축효과를 얻을 수 있을 것이다. 다만, 압축의 부담인 압축 시간이 요구되는데, 실제로 CPU와 RAM의 접근 속도가 플래시 메모리의 접근 속도 보다 월등하여 압축시 시간 부담이 크지 않았다. 압축된 노드를 검색할 경우에도 약간의 복원 시간이 필요하지만, 실험 결과 복원 시간이 크지 않아서 검색 성능에 별로 영향을 미치치 않았다.

<표 1>을 살펴보면, 플래시 메모리의 읽기 속도는 512 바이트 당 36μs이고 RAM의 읽기 속도는 2.6μs이므로, RAM은 플래시 메모리에 비하여 약 14배 정도 빠르다. 또한, 플래시 메모리의 쓰기 속도는 512 바이트 당 226μs이고 RAM의 읽기 속도는 2.6μs이므로, RAM은 플래시 메모리에 비하여 약 87배 정도 빠르다. 더욱이 플래시 메모리의 쓰기 연산은 매우 느린 소거 연산을 수반하므로 실제 쓰기 속도는 이 보다 더 느리다. 따라서 RAM의 접근 속도가 상대적으로 매우 빠르고 휴대용 컴퓨터의 CPU의

성능도 점차 개선되므로, 현재도 압축 부담이 크지 않으며, 향후 계속적으로 감소하게 된다.

본 실험에 사용된 압축 알고리즘은 단순하고 공개적으로 쉽게 구할 수 있는 LZO1X 압축 기법[LZO1X, 2006]이다. 이 알고리즘의 성능은 공개되어 있는데, PDA 200MHz 기준으로 보면, 압축 속도는 5MB/sec이고, 복원 속도는 20MB/sec이다. 본 실험에서 트리 샘플로서 압축률을 측정해보니 중간 노드는 55.6%이고, 리프 노드는 54.7%로 나타났다. 물론 이 보다 더 압축률이 좋은 알고리즘이 많이 있으므로, 추후에 더 성능을 개선할 수 있으나, 본 연구의 편의상 프로그램 코드가 공개되어 있는 LZO1X 압축 기법을 사용하였다.

본 연구에서 압축을 하는 이유는 한 노드에 많은 엔트리(키값과 포인터)들을 넣기 위함이 아니다. 오히려, 노드당 엔트리들을 많이 채워서 점유율을 높이면, 검색속도만 조금 개선될 뿐, 삽입 및 삭제시 빈번한 노드 이동으로 쓰기 연산이 증가하여, 전체적인 성능은 감소하게 되어 있다. 대신에 압축을 통하여 저장 공간을 줄여서, 앞부분에 1차 쓰기를 하고, 후에 트리 수정시에 뒷부분에 2차로 이어 쓰기를 하기 위함이다. 이러한 이어쓰기는 소거 연산의 수를 반으로 줄임으로써 전체적인 트리의 처리 성능을 높이게 된다. 다음은 압축이어쓰기 연산을 위하여 필요한 블록 헤더와 페이지 헤더의 기본 구조이다.

본 연구의 F-Tree에서 입출력의 효율을 위하여 한 페이지당 하나의 노드를 수용하도록 설계

```
struct Block_Header { // 소거 세그먼트 블록
BYTE        B_Status;       // B_FREE : 초기화 된 후 사용되지 않음,
                            // B_WRITING : 쓰기 작업 중
                            // B_FULL : 블록이 사용된 페이지로 가득 찼음,
                            // B_PREINVALID : 재활용 대상 블록으로 복사하는 중임,
                            // B_INVALID : 무효화된 상태
int         No_Erase        // number of erase operations, 지움 연산 카운터
etc ... // 기타, 오류정정코드, 불량 블록 여부 체크, 예약 블록, 각종 태그 등의 정보.
}

struct Page_Header { // 노드가 저장될 페이지
BYTE        P_Status;       // P_FREE : 초기화 되어 기록 가능,
                            // P_PREWRITE : 쓰기 작업 중이나 완료 상태는 아님.
                            // P_COMPLETE : 쓰기 작업이 완료된 비압축 페이지,
                            // _P_COMPRESSED : 1차 압축이 완료된 압축 페이지,
                            // _P_COMPRESSED2 : 2차 압축이 완료된 압축 페이지,
                            // P_PREINVALID : 재활용작업으로 다른 위치로 복사중,
                            // P_INVALID : 무효화된 상태, 삭제된 상태
short Comp1_end;        // 1차로 압축된 데이터의 마지막 위치 오프셋
short Comp2_end;        // 2차로 압축된 데이터의 마지막 위치 오프셋
etc ... // 기타, 페이지 번호, 오류정정코드, 각종 태그 등의 정보가 저장됨.
}
```

```
struct Internal_Node {
        short isLeaf; // 0: 중간 노드
        short num_of_children; //fan-out
        Node *pointer_to_children[MAX1];
          // 하위 노드로의 포인터 배열
        Key key[MAX1-1];
};
```

```
struct Leaf_Node {
        short isLeaf; // 1 : 리프 노드
        short num_of_object ; //fan-out
        Node *pointer_to_object[MAX2];
          // 하위 오브젝트로의 포인터 배열
        Key key[MAX2];
        struct Leaf_node *next;
          // 다음 리프 노드로의 포인터
};
```

하였다. 이 때 압축 노드의 메타 데이터도 같이 저장되는데, 4바이트 정도로 적으므로, 페이지 내부에 단편화로 발생된 잔여 공간에 저장가능하다. 또한 플래시 메모리의 페이지 마다 존재하는 여분 공간(spare area)을 활용하여도 저장가능하다. 즉, 여분 공간에 존재하한 보통 16바이트 또는 64바이트 크기의 부가 영역을 활용할 수 있다. 다음은 한 페이지 당 한 개로 수용되는 중간 노드나 리프 노드의 구조이다.

본 연구에서 제안하는 F-Tree와 기존의 B^+-Tree는 노드 구조체 측면과 이 노드를 대상으로 한 인덱스 연산 측면에서 차이가 있다. 먼저, 저장할 인덱스 노드의 구조체 측면에서 보면, 기본적으로 F-Tree는 B^+-Tree를 기반으로 하였으므로, B^+-Tree가 가지고 있는 기본 구조체를 포함하고 있다. 그러나 플래시 메모리 상에서 압축이어쓰기를 실행하기 위하여, 압축 및 복원을 관리하기 위한 제어 데이터의 구조가 추가적으로 필요하고, 실제 킷값 및 포인터 등의 데이터를 압축한 콘텐츠를 노드 상에 저장하기 위한 구조체가 필요하다. 또한, 인덱스 연산 측면에서 보면, 트리 노드에 삽입/삭제/검색 연산 수행시 기존 B^+-Tree의 기본 연산에 압축 및 복원 연산이 추가되고, 저장될 노드에 플래시 메모리에서만 가능한 이어쓰기 연산 수행 기능이 필요하다. 트리 노드에 대한 압축이어쓰기는 중간 노드와 리프 노드에서만 수행되며, 루트 노드는 너무 빈번히 접근되어 병목현상이 발생할 수 있으므로, 압축 저장하지 않는다.

4. 시뮬레이션 및 성능 평가

본 연구에서 제안된 F-Tree의 성능을 검증하기 위하여 컴퓨터 시뮬레이션을 수행한 후 그 실험 결과를 분석해 보았다. 실제 실험을 대신하여 컴퓨터 시뮬레이션 방식을 통한 모의실험을 선택한 이유는 다음과 같다.

플래시 메모리는 거의 표준화 되어 있는 디스크나 메모리와는 달리 다양한 방식과 종류가 존재한다. 생산 방식에 따라서 NAND 방식, NOR 방식 등의 유형으로 나뉘며, 동일한 방식이라도 삼성, 도시바 등 제조회사에 따라서 특징이 다르다. 또한, 메모리 용량 변화와 페이지 크기, 블록 크기 등의 변화와 같은 다양한 실험을 수행할 수 없다. 그리고 시스템 중간 과정에서 발생하는 페이징, 클리닝, 가비지 수집 부하, 캐싱, 프리패칭, 버퍼링, 카피 오버헤드 등의 다양한 간섭 요소가 존재하여 실제 알고리즘의 효과를 일관성 있게 측정하기가 어렵다. 그리고 플래시 메모리의 속도 개선을 위하여 고속고가의 저장장치인 SRAM을 결합하거나 버스, 콘트롤러 등의 결합으로 성능이 원래보다 상향된 형태도 많으므로 역시 일관된 측정에 어려움이 있다.

또한, 프로세서 수행 시간에 비하여 상대적으

로 느린 하드 디스크와는 달리 플래시 메모리는 빠른 수행 시간을 가지므로, 실제 시스템에서 실험 도중에 간섭되는 빈번한 이벤트의 발생, 커널의 다양한 인터럽트 처리, 멀티태스킹에 의한 프로세서의 문맥 교환(context switching)때문에 정확한 결과 값을 얻기가 어렵게 된다. 이와 같은 이유로 시뮬레이션을 사용하는 것이 더 융통성이 있으며 경제적이다[정재용, 2002].

본 시뮬레이션 수행시 F-Tree 색인 기법(FTR69)과 비교된 대상 색인은 가장 보편적으로 사용하는 B^+-Tree 기법(BTR69)이며, 전술한 바와 같이 트리 노드의 평균 점유율(69%)을 기본으로 하였다. 또한 트리가 거의 차 있는 경우(97%)의 B^+-Tree 기법(BTR97)도 포함하여, 총 세 가지 색인을 비교 시험하였다. 실험 도구는 CSIM [Schwetman, 1992] 시뮬레이션 언어와 비주얼 C++를 사용하였다. 실험이 수행된 하드웨어 환경은 펜티엄4-2.8 CPU와 메인 메모리 512M, 하드디스크 80G×2이며, 운영체제는 윈도우 2000 서버를 사용하였다.

4.1 실험용 트리 샘플의 구성

먼저, 실험용 샘플이 되는 트리가 필요한데, 현존하는 PDA가 수용할 수 있는 최대 크기를 기준으로 하여 실제 환경에 근접하도록 구성하였다. 입출력의 효율을 위하여 한 노드의 크기는 플래시 메모리의 한 페이지의 크기(512B)와 같도록 설정하였다. 노드를 포인팅하기 위한 포인터의 크기는 일반적인 포인터 크기인 4바이트에 플래시 메모리 식별자 1바이트를 추가하여 5바이트의 크기로 설정하였다. 또한, 키필드의 크기도 10바이트로 설정하였는데, 일반적인 정수, 실수 등의 데이터 형을 포함하여 키워드 정도의 문자열을 포함할 수 있는 여유가 되는 크기이다. 물론, 이러한 크기의 설정은 사용자의 작업환경에 따라서 변경될 수 있으나 실험의 목적과 결과의 순위에 크게 영향을 미치지는 않는다.

샘플용 트리의 크기는 기본적인 B^+-Tree 구조인데, 이 트리를 루트 노드부터 시작하여 리프 노드까지 구성하여, 플래시 메모리 256M~512M를 탑재한 보편적인 PDA에서 수용할 수 있는 적절한 크기를 산정해 보았다. 먼저 중간 노드의 크기는 한 페이지의 크기인 512B 이하로 할 때, 몇 개의 하위 노드를 포인팅 가능한지, 노드 포인터의 개수 p를 계산하였다. 자세한 식의 설명은 [황규영, 2000]에 나타나 있다.

$$(p \times P) + ((p - 1) \times V) \leq 512$$

여기서, 전술한 바와 같이, P는 포인터 크기로서 5 바이트이며, V는 키필드 크기로 10 바이트 이므로, 이 식을 정리하면,

$$(p \times 5) + (p - 1) \times 10 = 15p - 10 \leq 512$$

이며, p값은 정수로 34가 된다. 즉, 한 노드당 최대 34개의 포인터가 저장 가능하다. 여기서 한 노드당 69%의 평균점유율(avg fill factor)을 곱하면 적정한 포인터 개수는 23개가 되며, 따라서 키의 개수는 22개가 된다. 다음으로 리프 노드에는 몇 개의 데이터 포인터를 저장 가능한지, 노드 포인터의 개수 pl를 계산하였다.

$$pl \times (P + V) + P \leq 512$$

여기서 +P는 리프 노드에서 순차 접근을 위한 다음 노드로의 포인터 크기이다. 이 식을 정리하면,

$$pl \times (5 + 10) + 5 \leq 512$$

이며, pl 값은 정수로 33이 된다. 즉, 한 노드

당 최대 33개의 포인터가 저장 가능하다. 여기서 한 노드당 69%의 점유율을 곱하면 적정한 포인터 저장 개수는 23개가 되며, 키 개수도 23개가 저장된다. 이 계산 결과를 바탕으로 일반 PDA에서 수용 가능한 범위 내에서 루트 노드에서부터 노드를 추가하여 B^+-Tree를 구성해 보면 아래의 <표 2>와 같다.

〈표 2〉 실험용 트리의 샘플의 구성

노드 타입	레벨	노드수	키 개수	포인터수
루트 노드	0	1	22	23
중간 노드	1	23	506	529
중간 노드	2	529	11,638	12,167
리프 노드	3	12,167	279,841	279,841

이 실험용 트리에는 총 12,720개의 노드가 존재하며, 총 6.51 메가바이트의 색인 공간이 필요하다. 또한, 이 색인이 포인트 하는 데이터는 리프 노드의 포인터 개수인 279,841개가 되며, 차지하는 총 페이지 공간은 143.3 메가바이트가 된다. 따라서 색인과 데이터 크기를 합하면, 총 150 메가바이트가 저장된다. 실제 PDA에 256M나 512M의 플래시 메모리를 탑재하더라도, 다양한 소프트웨어와, 사용자 파일, 동영상, MP3 파일 등이 존재하므로, 150 메가바이트의 트리 샘플은 실제로도 적지 않는 크기라고 생각된다. 만일, 한 레벨 더 트리를 확장하여 레벨 4까지 계산해 보면, 3기가바이트가 넘게 저장되므로, 현재의 PDA에서 저장하기에는 아직은 비현실적인 크기가 된다. 따라서 레벨3까지 적용한 트리를 실험용 샘플로 사용하였다.

4.2 모의실험 환경의 구성

다음으로 플래시 메모리 시스템을 위한 시뮬레이션 모듈이 필요한데, 사용된 모델은 CSIM에서 제공되는 폐쇄형 큐잉 모델(Closed Queuing Model)이다. 이 모듈을 통하여 초당 일정한 수의 읽기 쓰기 트랜잭션을 생성하고, 이와 연관된 데이터가 검색 또는 수정되면서 발생한 F-Tree 색인의 처리 시간과 성능을 측정하면 된다.

플래시 메모리 데이터베이스 운영 환경을 위한 시뮬레이션의 주요 성능 평가 지표는 트랜잭션 처리치(transaction throughput)와 응답시간(response time)이다. 트랜잭션 처리치는 초당 몇 개의 트랜잭션이 처리되었는지를 의미하고, 응답시간은 트랜잭션이 발생한 후 수행까지의 지연 시간을 의미한다.

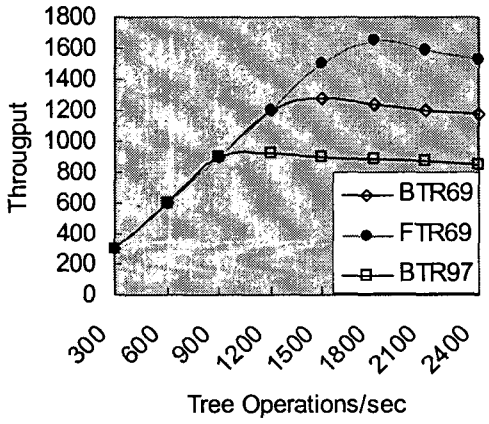
주요 시뮬레이션 파라미터는 초당 생성된 트랜잭션의 수, 읽기 연산 수행시간, 쓰기 연산 수행 시간, 소거 연산 수행시간 등이다. 초당 생성된 트랜잭션의 수는 300개에서부터 300개 단위로 2,400개까지 변화시켜 보았으며, 이는 시뮬레이션 시스템에 가해지는 작업 부하를 의미한다. 플래시 메모리의 읽기 연산 수행 시간은 36 μs로 설정하였고, 쓰기 연산 수행 시간은 266μs로 설정하였으며, 소거 연산 수행 시간은 2ms로 설정하였다. 각 연산 수행 시간은 기존의 연구[임근수, 2003]에서 제시한 연산 수행에 필요한 실측치이다. 위의 설정된 주요 파라미터 외의 부수적인 파라미터의 수치는 고정된 값이지만 실험 진행 과정에 필요시 변화시킬 수 있게 하였다.

4.3 실험 결과 및 분석

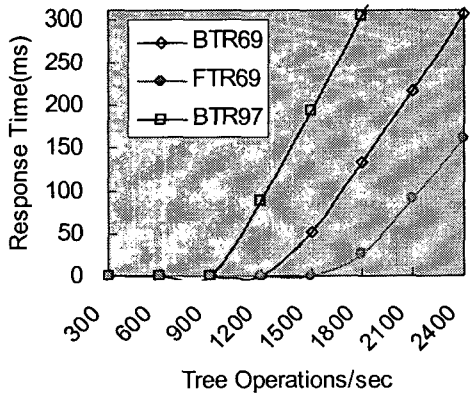
시뮬레이션은 한 노드에 대한 평균 점유율 69%로 B^+-Tree를 구성한 BTR69 색인과 이 점유율을 97%로 변형한 BTR97 색인과 제안하는 F-Tree 구성인 FTR69 색인을 대상으로 수행하였으며, 트리에 대한 삽입, 삭제 부하에 따른

각 색인의 처리 성능과 처리 시간을 분석하였다.

실험은 기본적으로 플래시 메모리를 탑재한 정보 기기에서 부과한 시스템 부하, 즉 초당 발생된 트리 연산의 수가 색인들의 성능에 어떤 영향을 미치는 지를 분석하기 위함이다. 그림4는 초당 발생된 트리 연산의 수의 증가에 따른 처리치를 표시한 그래프이다. 발생된 초당 트리 연산의 수가 늘어날수록 점차로 연산의 처리 결과치가 증가함을 알 수 있다. 또한, 전반적인 트리 연산의 처리 성능을 측정한 결과, FTR69가 BTR69와 BTR97 보다 높게 나타났다.



〈그림 4〉 초당 트리 연산 처리치의 비교



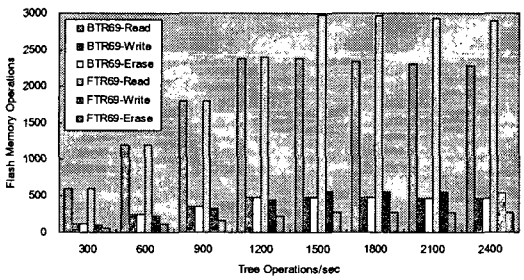
〈그림 5〉 트리 연산의 응답 시간의 비교

<그림 4>에서 보면, 초당 트리 연산의 수가 대략 1,500개를 넘으면서 각 색인의 성능이 점점 낮아진다. 이는 초당 트리 연산수의 증가에 의한 플래시 메모리 연산 집중화가 성능에 영향을 크게 미치는 주요 요소임을 의미하며, 이 수치 이상으로 트랜잭션을 활성화시키는 것이 성능 향상에 도움이 되지 않음을 의미한다. 특히 쓰기 연산의 느린 속도와 수반 되는 소거 연산이 주요 요인이다. 그래프에서 보면 연산의 부하가 적은 시작 구간에서는 느린 쓰기와 소거 연산이 시스템 안에서 수용되어 성능 저하를 일으키지 않는다. 하지만, 시작 구간을 지나면서 생성되는 트리 연산의 수가 증가하므로, 트리의 삽입 및 삭제 연산, 즉 플래시 메모리에 쓰기 연산의 수도 증가하게 된다. 따라서 중간 구간 이후에서는 연산들이 작업 큐에 과도하게 쌓이면서, 연산 처리가 지체되며 결과적으로 성능이 서서히 저하되게 된다.

하지만, 동일한 조건에서도 제안한 FTR69 색인이 BTR69나 BTR97 색인에 비하여 성능이 상대적으로 더 높다. 트리 연산의 처리치가 감쇄되는 지점도 FTR69가 1800으로 BTR69의 1,500보다 나중에 발생하며, 전체 구간에서 FTR69가 BTR69보다 17% 정도 처리 성능이 높다. 특히, 최대 연산 처리치는 FTR69가 1,650개로 BTR69의 1,270개 보다 29.9% 정도 더 높다. 그 이유는 위의 색인 성능 저하의 주요 원인인 쓰기 연산의 부담을 FTR69 색인은 압축이어쓰기를 통하여 감소시킬 수 있었기 때문이며, 압축 및 복원에 대한 부담이 그리 크지 않았기 때문이다. 또한, 쓰기 연산수의 감소에 따라서, 플래시 메모리의 수명이 연장되며, 소거 연산을 위한 전력 소모도 줄이는 부수적인 효과도 기대할 수 있다.

또한, <그림 5>에서 트리 연산의 응답시간을

비교해보면, FTR69, BTR69, BTR97 순으로 우수하게 나타났다. 특히, 전체 구간에서는 FTR69가 BTR69보다 2.53배의 개선된 응답 속도를 나타내었다. 이 결과치로부터 트리 연산에 수반된 플래시 메모리의 쓰기와 소거 연산수의 감소가 처리 시간 개선 효과를 주고, 이는 다시 초당 트리 연산의 처리 성능을 개선시킴을 확인할 수 있다. 참고로 그림4에서 BTR97 색인이 BTR69보다 성능이 낮은 이유는 BTR69에 비하여 노드에 조밀하게 색인 정보를 저장하여, 전체적인 검색 노드수가 줄어들어, 검색 연산에서는 장점이 있는 반면에, 노드에 대한 삽입 및 삭제 연산에 수반되는 쓰기 연산이 상대적으로 빈번하게 발생하였기 때문이다.



〈그림 6〉 입출력 연산수의 비교

<그림 6>은 본 제안 기법이 트리 연산의 처리치와 응답시간 측면에 대해서는 좋은 성능을 보이는 결과의 기본 원인이라 할 수 있는 플래시 메모리에서의 읽기 연산, 쓰기 연산, 소거 연산량의 변화를 나타낸 그래프이다. 이 그래프에서 x축에서 초당 트리 연산의 수가 증가함에 따라서, y축의 이 세 가지 연산별 처리량도 증가하게 된다. 전체 구간에서 BTR69보다 FTR69의 읽기 연산 및 쓰기 연산 처리량이 더 많다. 이 사실은 <그림 4>에서 BTR69보다 FTR69가 초당 트리 연산 처리치가 더 높은 이유를 뒷받침한다. 반면에, 소거 연산량을 비교하여 보면, 전체 실험 구간에서 BTR69보다 FTR69가 훨씬 더 적은 수의 소거 연산을 수행한 것을 알 수 있다. <그림 6>의 세 번째 막대그래프의 BTR69-Erase와 마지막의 FTR69-Erase의 연산수를 비교해보면 최대 두 배 정도의 차이가 난다. 그 이유는 본 제안 기법의 압축이어쓰기 효과로 인한 것이며, 이 효과에 의한 느린 소거 연산량의 감소는 전체적인 시스템의 성능 향상에 기여하게 된다.

5. 결론 및 향후 과제

본 연구에서는 휴대형 정보기기의 데이터 저장 장치로 많이 사용되는 플래시 메모리와 관련하여 기존의 트리 기반의 색인 저장 기술을 분석하였다. 그리고 트리 색인에서 발생되는 빈번한 검색과 삽입, 삭제 연산에 대하여 처리 성능과 응답 성능을 더욱 개선할 수 있는 F-Tree 색인 기법을 제안하였다. F-Tree 색인 기법에서는 쓰기 연산과 소거 연산이 상대적으로 매우 느린 플래시 메모리 입출력의 단점을 고려하여 새로운 압축 이어쓰기 기법을 활용하였으며, 그 결과 저속의 소거 연산의 실행 횟수를 크게 낮추었다. 이로부터 트리 연산에 대한 응답 성능과 처리 성능이 향상되었다.

제안 기법의 효과를 검증하기 위한 시뮬레이션과 분석 결과, 작업 부하가 적은 시작 구간에서는 기존 색인 기법과 비슷하게 나타났지만, 트리 연산이 증가하여 노드의 입출력 부하가 증가하면 기존 색인 기법에 비하여 더 우수한 성능이 나타났다. 전체적인 실험 구간에서 F-Tree는 기존 색인 기법에 비하여 평균 2.53배의 개선된 응답 성능을 보였으며, 30%의 개선된 트리 연산의 처리 성능을 보였다. 현재, 휴대용 정보 시스템의 데이터 저장장치로서 대부분 플래시 메모리가 사용된다는 측면에서, 플래시 미디

어 기반 데이터 처리 기술은 향후 지속적인 연구가 요구된다.

참 고 문 헌

[1] 이옥희, 김진호, 차재혁, "스페어 영역을 활용한 NAND 플래시 메모리 관리", *정보처리학회 추계학술발표대회 논문집*, 제11권 제2호, 2004, pp. 149-152.

[2] 이창우, 안경환, 홍봉희, "이동체 데이터베이스를 위한 메인 메모리 색인의 성능 결정 요소에 관한 연구", *정보처리학회 춘계 학술대회 논문집*, 제10권 제1호, 2003, pp. 1575-1578.

[3] 임근수, 고건 "플래시 메모리 기반 저장장치의 설계 기법", *정보과학회 추계 학술대회*, 제30권 제2-1호, 2003, pp. 274-276.

[4] 정재용, 노삼혁, 민상렬, 조유근, "플래시 메모리 시뮬레이터의 설계 및 구현", *한국정보과학회 논문지 C-컴퓨팅의 실제*, 제8권 제1호, 2002, pp. 36-45.

[5] 황규영, 홍의경, 음두헌, 박영철, 김진호, "*데이터베이스 시스템*", 생능출판사, 2000.

[6] Beckmann N., H. P. Kriegel, R. Schneider, and B. Seeger, "The R*Tree : An Efficient and Robust Access Method for Points and Rectangles", *Proc. of ACM SIGMOD Intl. Symp. on the Management of Data*, 1990, pp. 322-331.

[7] Cha S. K., J. H. Park, and B. D. Park, "Xmas : An Extensible Main-Memory Storage System", *Proc. of 6th ACM Int'l Conference on Information and Knowledge Management*, Nov. 1997.

[8] Chanik Park, Jaeyu Seo, Dongyoung Seo, Shinhan Kim, and Bumsoo Kim, "Cost-Efficient Memory Architecture Design of NAND Flash Memory Embedded Systems", *21st International Conference on Computer Design*, San Jose, California, October 13-15, 2003, pp. 474-479.

[9] Chin-Hsien Wu, Li-Pin Chang, and Tei-Wei Kuo, "An Efficient R-Tree Implementation over Flash-Memory Storage Systems", *Proc. of ACM GIS'03*, New Orleans, Louisiana, USA, November 7-8, 2003a, pp. 17-24.

[10] Chin-Hsien Wu, Li-Pin Chang, and Tei-Wei Kuo, "An Efficient B-Tree Layer for Flash-Memory Storage Systems", *Proc. RTCSA*, Tainan, Taiwan, 2003b, pp. 409-430.

[11] Eui-deok Hwang and Jae-Hyuk Cha, "A Recovery Mechanism applying the Shadow-Paging technique to Flash Memory Base LFS", *Proc. of KISS Fall*, Vol. 31, No, 2, pp. 199-201.

[12] Hongjun Lu, Yuet Yeung Ng, and Zengping Tang, "T-Tree or B-Tree : Main Memory Database Index Structure Revisited", *Proc. of 11th Australasian Database Conference*, 2000.

[13] Hyung Gyu Lee and Naehyuck Chang, "Energy-Aware Memory Allocation in Heterogeneous Non-Volatile Memory Systems", *ACM ISLPED'03*, Seoul, Korea, August 25-27, 2003, pp. 420-423.

[14] JungHyun Nam and Dong-Joo Park, "The Efficient Design and Implementation of The B-Tree on Flash Memory", *Proc. of KISS Fall*, 2005, Vol. 32, No. 2, pp. 55-57.

[15] Keun Soo Yim, Jihong Kim, and Kern Koh, "A Fast Start-Up Technique for Flash Memory Based Computing Systems",

2005 ACM Symposium on Applied Computing, Santa Fe, New Mexico, USA, March 13-17, 2003, pp. 843-849.

[16] Lehman T. J. and M. J. Carey, "A Study of Index Structures for Main Memory Database Management Systems", *Proc. of 12th Intl. Conf. on Very Large Database*, 1986, pp. 294-303.

[17] Li-Pin Chang and Tei-Wei Kuo, "An Efficient Management Schemefor Large-Scale Flash-Memory Storage Systems", *Proc. of ACM SAC'04*, Nicosia, Cyprus, March 14-17, 2004, pp. 862-868.

[18] Schwetman H., *CSIM User's Guide for Use with CSIM Revision 16*, Microelectronics and Computer Technology Corporation, 1992.

[19] Samsung, What is Flash?, http://www.samsung.com/Products/Semiconductor/Flash/WhatisFlash/FlashStructure.htm, 2006.

[20] LZO1X, http://www.oberhumer.com/open-source/lzo/#download, 2006.

□ 저자소개

변 시 우

연세대학교 전산과학과를 졸업하고, 한국과학기술원에서 전산학 석사와 박사 학위를 취득하였다. 현재 안양대학교 디지털미디어 공학과의 부교수로 재직하고 있으며, 주요 관심분야는 모바일 컴퓨팅, 분산데이터베이스, 휴대용 데이터베이스, 전자상거래 분야이다.

◈ 이 논문은 2006년 08월 14일 접수하여 1차 수정을 거쳐 2006년 12월 11일 게재확정되었습니다.