내장형 정보기기를 위한 플래시 메모리 기반 색인 기법

Flash Memory based Indexing Scheme for Embedded Information Devices

변시우*, 노창배**, 허문행***
Siwoo Byun, Changbae Roh, Moonhaeng Huh

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.

Key Words : embedded database, tree indexing, portable devices, flash memory

1. 서 론

각종 소형 정보기기들이 대중화됨에 따라, 정보 저장용 미디어로 플래시 메모리가 보편적으로 활용되게 되었다. 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 특성들이 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 접근 시에 별 문제가 없다.[1] 하지만, 플래시 메모리의 쓰기 연산의 경우에는 속도가 읽기 연산 대비 20배 정도의 많은 시간이 소모되어 매우 느리며, 메인 메모리처럼 Update-In-Place가 불가능한 Update-Out-Place 구조이다.[2] 더욱이 쓰기 연산 전에 2㎳ 정도의 느린 속도로 소거 연산을 반드시 수행해야 하므로 연산의 총 부담이 크게 늘어난다.

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.

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.[3] 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.[4] 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.

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 들어갈 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.[5] 주로 B-Tree, B$^+$-Tree[6] 계열이 많이 사용되며, 공간 색인으로는 R-Tree, R*-Tree[7] 계열이 사용되고 있다.

메인 메모리 기반 색인 및 저장 시스템은 디스크 기반 시스템에 비하여 디스크에 접근하는 시간이 대폭 줄어들기 때문에 훨씬 더 빠른 성능을 보여줄 수 있다. 일반적으로 T-Tree가 1차원 데이터를 위한 색인으로서 좋은 성능을 보이며, 비교적 적합하다고 한다. T-Tree는 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.[8] 그러나 [9]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 구조를 개선한 B-Tree 계열의 인덱스가 많이 사용된다.

그리고 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.[8]

그러나 플래시 메모리 기반 저장 시스템에서는 기존의 인덱스 구조와 관리 기법을 그대로 적용할 수 없다. 그 이유는 다음과 같다.

① 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[2]. 그러므로 쓰기 연산을 판독 연산과 같은 비중으로 적용할 수 없다.

② 쓰기 연산의 전력소모량은 읽기 연산의 9배 정도가 되므로 배터리 용량이 매우 제한적인 휴대용 정보기기에 9배 정도의 큰 전력 부담을 준다.

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

3. 플래시 메모리 기반 색인 저장 및 관리 기법 제안

3.1 제안 목표

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.[9] 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 및 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.

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.[10] 따라서 플래시 메모리 상에 B-Tree 계열의 색인 구축할 때 중요한 문제는 쓰기 연산을 최대한 줄이는 것이다.

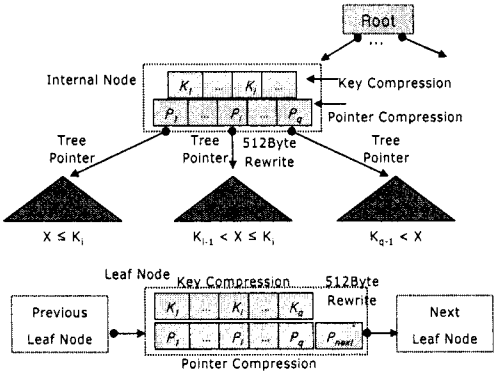
기본적으로 B-Tree는 하위 노드에 대한 포인터 정보와 함께 데이터 관련 정보를 한 노드 안에 보관한다. 그러나 B^+-Tree는 중간 노드에 데이터 관련 정보를 넣지 않고 리프 노드에 이를 저장한다. 즉, 중간 노드에서는 순수한 색인 정보만을 저장하므로, 색인 자체의 저장 효율이 높아진다. 또한, 리프 노드에서는 색인 검색의 도움 없이 바로 순차적인 접근이 가능한 장점도 있다. 따라서 플래시 메모리의 색인으로서는 기본적인 B-Tree 보다도 B^+-Tree가 더 적합하다.

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 [6]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 초기화하는 소거 연산을 먼저 수행하여야 한다. 그러나 1에서 0으로의 쓰기 연산은 소거 연산의 수행 없이도 가능한 특수한 성질이 있다.[11] 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-ROM에

서, 저장할 용량이 작을 경우 먼저 일부 쓰기를 수행한 후, 나중에 추가 데이터가 발생하면, 나머지 빈 공간에 다시 이어 쓰기를 하는 것과 유사하다. 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 1은 제안된 F-Tree의 중간 노드와 리프 노드의 구조이다.



(그림 1) F-Tree의 중간 노드와 리프 노드의 구조

노드 압축은 일반적인 압축 알고리즘을 적용하되, 압축 효율을 높이기 위하여, 키 부분과 포인터 부분을 분리하여 압축한다. 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 미치지 않았다.

본 실험에 사용된 압축 알고리즘은 단순하고 공개적으로 쉽게 구할 수 있는 LZO1X 압축 기법[12]이다. 이 알고리즘의 성능은 공개되어 있는데, PDA 200MHz 기준으로 보면, 압축 속도는 5 MB/sec이고, 복원 속도는 20 MB/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.

본 연구의 F-Tree에서 입출력의 효율을 위하여 한 페이지당 하나의 노드를 수용하도록 설계하였다. 이 때 압축 노드의 메타 데이터도 같이 저장된다.

4. 결론 및 향후 과제

최근 소형 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 낮추었다. 이로부터 트리구조의 색인 연산에 대한 응답 성능과 처리 성능이 향상된다.

참 고 문 헌

[1] 이옥희, 김진호, 차재혁, "스페어 영역을 활용한 NAND 플래시 메모리 관리", 정보처리학회 추계학술발표대회 논문집, 제 11권 2호, pp. 149-152, 2004.11.

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

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

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

[5] 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.

[6] 황규영, 홍의경, 옴두현, 박영철, 김진호, "데이터베이스 시스템", 생능출판사, 2000

[7] Beckmann N., H. P. Kriegel, R. Schneider, 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, pp. 322-331, 1990.

[8] 이창우, 안경환, 홍봉희, "이동체 데이터베이스를 위한 메인 메모리 색인의 성능 결정 요소에 관한 연구", 정보처리학회 춘계 학술대회 논문집, 제10권 1호, pp. 1575-1578, 2003.5.

[9] 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

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

[11] 정재용, 노삼혁, 민상렬, 조유근, 플래시 메모리 시뮬레이터의 설계 및 구현, 한국정보과학회 논문지 C-컴퓨팅의 실제, 제 8권 1호, pp. 36-45, 2002년 2월.

[12] LZO1X, http://www.oberhumer.com/opensource/lzo/#download, 2006